Universidad
Carlos III de Madrid

TRABAJO FIN DE GRADO

Diseño e implementación de un sistema de monitorización de temperatura capaz de comunicar de manera inalámbrica con un dispositivo móvil

Autor:

Adrián Paredes Intriago

Tutor:

Jonathan Crespo Herrero

Grado en Ingeniería Electrónica Industrial y Automática

Madrid, Septiembre de 2014

Agradecimientos

A mis padres, Juan y Raquel.

A mi hermana, Claudia.

Por no rendirse jamás.

Resumen

En este trabajo se desarrolla un sistema de monitorización de temperatura capaz de comunicar de manera inalámbrica con un dispositivo móvil. El sistema está compuesto por un microcontrolador, un sensor de temperatura y un módulo Bluetooth. Las comunicaciones entre los componentes del sistema son llevadas a cabo a través de los buses de comunicación en serie SPI e I^2C, mientras que las comunicaciones entre el sistema y el dispositivo móvil se realizan a través del protocolo de comunicación inalámbrica Bluetooth de bajo consumo, también conocido como BLE. En cuanto al dispositivo móvil, todas las comunicaciones con el sistema son gestionadas mediante una aplicación Android. Esta aplicación muestra las mediciones del sistema en tiempo real permitiendo a su vez configurar varios tipos de notificaciones y alarmas.

Palabras clave: Sistema de monitorización de temperatura, dispositivos móviles, Bluetooth de bajo consumo, Android.

Abstract

This work develops a temperature monitoring system able to communicate in a wireless manner to a mobile device. The system consists of a microcontroller, a temperature sensor and a Bluetooth module. The communications between the components of the system are done through the serial buses SPI and I^2C, whereas the communications between the system and mobile device are done through Bluetooth low energy, also known as BLE. On the mobile device side, all the communications with the system are handled through an Android application. This application shows real-time temperature measurements while permits to configure a wide variety of notifications and alarms.

Keywords: Temperature monitoring system, mobile devices, Bluetooth low energy, Android.

Índice de contenido

Índice de figuras

Índice de tablas

1 Introducción

El presente documento describe el proceso de creación e implementación de un sistema de medición de temperatura capaz de comunicarse de manera inalámbrica con un dispositivo móvil (p. ej. smartphone). Este sistema consta de un microcontrolador, un sensor y un módulo de Bluetooth. Gracias a este diseño modular es posible implementar diferentes aplicaciones variando el número y el tipo de sensores utilizados. De esta manera, aunque el sistema descrito en este documento utilice un sensor de temperatura, la solución desarrollada podrá extrapolarse a cualquier tipo de sensor que permita una comunicación I^2C con el microcontrolador.

1.1. Motivación y objetivos

El campo de las comunicaciones inalámbricas brinda múltiples posibilidades de conectividad entre dispositivos y sus aplicaciones han experimentado un gran auge en los últimos tiempos, gracias en parte a la expansión masiva y a escala mundial de los dispositivos móviles o smartphones. Por ello, una de las motivaciones a la hora de afrontar este proyecto ha sido comprender y poder desarrollar una aplicación que saque partido de las posibilidades de conexión inalámbrica que ofrecen los dispositivos móviles en la actualidad, explorando los conceptos de dispositivo que es capaz de interactuar con diferentes periféricos y del denominado Internet de las cosas o IOT (Internet of Things) [1].

Esta idea de conectividad total en la palma de la mano es la que ha impulsado la elección y posterior desarrollo de este proyecto en el cual, utilizando un sensor

relativamente básico como pueda ser un sensor de temperatura, se puede explorar y experimentar con el control del mismo de manera rápida y cómoda, mediante cualquier dispositivo móvil (teniendo en cuenta las restricciones relativas al hardware y al software).

De esta manera, los objetivos de este trabajo abarcan aspectos tales como la realización de las comunicaciones entre microcontrolador y sensor, modulo Bluetooth y microcontrolador mediante buses de comunicación en serie (I^2C y SPI), programación de una aplicación para el sistema operativo Android que gestione la comunicación entre dispositivos móviles y el sistema (la cual se realizará mediante la tecnología Bluetooth low energy), manejo del hardware y software necesarios, análisis de alternativas al diseño, etc.

2 Estado del arte

El sistema planteado, al igual que la mayoría de sistemas de este tipo, requiere un control inteligente, y en este caso es proporcionado por un microcontrolador. Este será utilizado como parte central del diseño, funcionando entre el sensor de temperatura y el módulo Bluetooth, controlando cuándo y cómo son utilizados. Es por ello que una breve introducción sobre su historia y evolución puede resultar relevante y por ello se incluye en este documento. Además, se analizará la tecnología Bluetooth de bajo consumo [2], o Bluetooth low energy, la cual se utilizará en las comunicaciones entre el sistema y el dispositivo móvil.

2.1. Microcontroladores

Evolución e historia de los microcontroladores

Durante la segunda mitad del siglo XX se pudo asistir a lo que podría denominarse un cambio de escenario, en el cual se materializaron avances que sentaron las bases de la electrónica actual.

Si hubiese que escoger cuales han sido los avances más significativos en el campo de la electrónica de los últimos 100 años uno de ellos sería sin duda el que tuvo lugar en Diciembre de 1947, cuando el primer transistor bipolar fue construido en los Laboratorios Bell por John Bardeen, Walter Houser Brattain y William Bradford Shockley, los cuales fueron galardonados con el Premio Nobel de Física en 1956 por el

descubrimiento [3]. Saltando a 1958, fecha en la que Jack Kilby construyo el primer circuito integrado y de ahí a 1970, año en el cual surgió el primer microprocesador de la historia, el Intel 4004. Dichas fechas no han sido escogidas al azar y cada una de ellas tiene una importancia fundamental en la aparición de los microcontroladores.

Se puede observar cómo, a pesar de toda su versatilidad, los procesadores tenían y tienen un hándicap a la hora de implementar según que sistemas, ya que precisan de circuitos integrados adicionales tales como memorias RAM, memorias ROM, circuitos integrados para los puertos de entrada y salida, etc.
La solución no tardaría en llegar, y en 1971 Gary Boone y Michael Cochran de Texas Instruments diseñaron algo que eliminaba la necesidad de circuitería adicional integrando todo lo necesario en un solo chip, el TMS 1000, había nacido el primer microcontrolador (Figura 1) [4]. A diferencia de los microprocesadores de Intel, el microcontrolador de 4 bits de Texas Instruments no fue sacado al mercado inmediatamente, siendo utilizado en una calculadora de la misma compañía que salió a la venta en 1972.

Durante años, Texas Instruments perfeccionó el diseño y finalmente el chip salió al mercado en 1974, siendo producido en 40 diferentes versiones al precio de 2 dólares. Las ventas superaron los 100 millones de unidades y el chip fue utilizado entre otras aplicaciones en juguetes, alarmas antirrobo, fotocopiadoras, tocadiscos, juegos y, como no, calculadoras.

Después del TMS 1000 surgieron otros microcontroladores como el Fairchild F8 de 8 bits fabricado en dos chips por Fairchild Semiconductor o su versión en un único chip, el MK3870, fabricado por Mostek. Otros fabricantes como Motorola o Rockwell también empezaron a sacar sus propios microcontroladores, así como Intel que, a pesar de reaccionar tarde (1976), saco al mercado su primer microcontrolador, el Intel 8048. Al 8048 le siguió en 1980 el 8051, siendo aún producidas variaciones del mismo hoy en día que son utilizadas en multitud de dispositivos de electrónica de consumo.

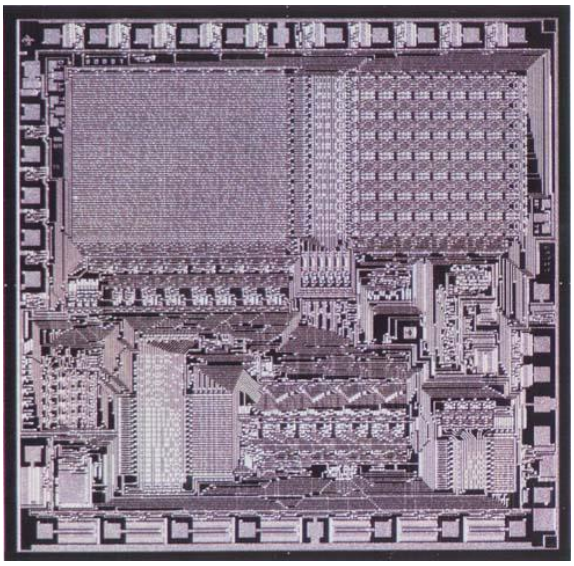


Figura 1. TMS 1000. En la figura puede apreciarse dos cuadros de mayor tamaño, el situado a la izquierda corresponde a 1K bits de ROM y el de la derecha a 256 bits de RAM. Los otros circuitos son los encargados de realizar los cálculos. Tamaño real: 3.098 x .3.632 mm.

Respecto al uso de memorias, parte fundamental del microcontrolador, los microcontroladores más primitivos se adaptaron rápidamente al uso de memorias EPROM, teniendo que esperar dos décadas más, hasta 1993, para empezar a ver la implementación de memorias EEPROM, fácilmente manipulables eléctricamente y que sustituirían rápidamente a las antiguas EPROM, más baratas pero con visibles desventajas frente a su versión evolucionada [5]. Poco después la compañía Atmel lanzo el primer microcontrolador que utilizaba memoria Flash, siendo estas en la actualidad las más utilizadas, ya que permiten velocidades de funcionamiento muy superiores a las de las memorias EEPROM.

Hoy en día, la producción de microcontroladores se cuenta por miles de millones al año y están presentes no solo en sistemas electrónicos sino en casi cualquier máquina, dispositivo, sistema o equipamiento utilizado en aplicaciones tan dispares como pueda ser la agricultura o en la fabricación de vehículos eléctricos. Frente a los primeros microcontroladores de 4 bits, la estrella actual del mercado son los microcontroladores de 32 bits, con el fabricante ARM y sus Cortex-M a la cabeza [6].

2.2. Bluetooth low energy

El concepto de un sensor, de temperatura en este caso, que comunique mediante Bluetooth con un dispositivo móvil no es algo nuevo ni rompedor, existiendo alternativas ya disponibles en el mercado [7] [8]. Sin embargo, el número de ellos que en la actualidad utiliza la tecnología Bluetooth de bajo consumo, o Bluetooth low energy (en adelante BLE) aún es reducido, a pesar de las múltiples y notorias ventajas que esta tecnología ofrece frente al Bluetooth clásico para ciertas aplicaciones. En las siguientes páginas se analizará cómo funciona esta tecnología, así como ciertos aspectos o características de la misma que pueden ayudar a comprender porque en los próximos años se podrá ver un aumento considerable del número de dispositivos que la utilicen en sus aplicaciones [9].

2.2.1 Evolución de la tecnología Bluetooth

Desarrollado por el Bluetooth Special Interest Group (SIG), la tecnología Bluetooth low energy fue introducida como un subconjunto en la especificación Bluetooth 4.0 [2]. Con una pila de protocolo completamente nueva para desarrollar rápidamente enlaces sencillos, se continuo con la misma en la especificación 4.1 oficialmente anunciada el 4 de Diciembre de 2013, mejorando no tanto el hardware, el cual se mantuvo invariable, sino el software existente. En contraste con el Bluetooth clásico, el BLE ha sido diseñado como una solución de bajo consumo para aplicaciones de control y monitorización, surgiendo como alternativa a otras tecnologías inalámbricas de bajo consumo ya existentes como puedan ser ZigBee, Z-Wave o 6LoWPAN.

Dentro de los dispositivos compatibles con la tecnología BLE, el SIG Bluetooth diferencia entre dos tipos de dispositivos: Aquellos que permiten un uso tanto del Bluetooth clásico como del BLE son denominados Bluetooth Smart Ready, permitiendo un uso dual de la tecnología Bluetooth existente. Por otro lado los dispositivos que solo son compatibles con la tecnología BLE (single mode o modo único) incluyendo únicamente la pila de protocolo de baja energía son denominados Bluetooth Smart.

Desde 1999, año en el que el estándar Bluetooth para comunicaciones inalámbricas fue creado, convirtiéndose en estándar IEEE 802.15.1 en 2002, los dispositivos de bajo

consumo, como ratones para PC, monitores cardiacos, dispositivos de seguridad en el hogar,… han ido disponiendo de diferentes alternativas entre las que elegir para realizar las comunicaciones inalámbricas, como por ejemplo el anteriormente citado Zigbee. Teniendo en mente estas aplicaciones de muy baja potencia la versión de bajo consumo de Bluetooth fue creada, con el objetivo de poder funcionar durante por lo menos un año con una simple pila de botón. Este consumo tan reducido es posible como se podrá ver más adelante gracias, entre otros aspectos, al envío de pequeños paquetes (23 bytes MTU, o lo que es lo mismo 23 bytes Maximum Transfer Unit) y a la reducción de comunicaciones tanto en cantidad como en duración de las mismas.

En la tabla 1 se puede ver la fecha en la que fueron introducidas las diferentes especificaciones Bluetooth y sus características más destacables.

Nomenclatura	Fecha de lanzamiento	Características
BR	**1.1 (2002)**	Basic Rate (1 Mbit/s)
EDR	**2.0 (2004)**	Enhanced Data Rate (2 y 3 Mbit/s)
HS	**3.0 (2009)**	High Speed (24 Mbit/s)
LE	**4.0 (2010)**	Low Energy (1 Mbit/s ultra low power)
Bluetooth Smart	**4.0 (2011)**	Single-mode, LE-only radio
Bluetooth Smart Ready	**4.0 (2011)**	Dual-mode, BR/EDR and LE dual radio

Tabla 1: Especificaciones Bluetooth en orden cronológico. Fuente: Nordic Semiconductor

2.2.2 Arquitectura Bluetooth low energy

Con el fin de entender el funcionamiento de la tecnología BLE se debe analizar primero como está diseñada y cuál es su arquitectura. Por ello se accederá a la especificación del sistema Bluetooth 4.0 [2] en la cual fue introducida.

La especificación principal de Bluetooth (denominada core) define el nivel físico (PHY) y el control de acceso al medio (MAC) de una red inalámbrica de área personal, en este

caso Bluetooth. La especificación principal define el sistema básico, pero su diseño potencia la flexibilidad. Por ello, hay multitud de opciones, definidas por los perfiles Bluetooth en especificaciones complementarias. El sistema básico está formado por un transceptor de radiofrecuencia, el nivel de banda base y la pila de protocolos Bluetooth, y otorga conectividad a todo un rango de dispositivos.

Según la especificación 4.0 para que un dispositivo Bluetooth pueda ser catalogado como “Low Energy” debe cumplir (tabla 2):

-En la parte “Host” o Anfitrión: Para las capas GAP, ATT, GATT y SM todas las características obligatorias y para la capa L2CAP todas las características obligatorias relacionadas con el canal de señalización low energy (CID 0x0005).

-En la parte “Controller” o Controlador: Para las capas PHY y LL (physical y link layer) todas las características obligatorias.

Host Part:

Layer	Required Features
L2CAP ([Vol. 3] Part A)	L2CAP LE Signaling Channel (CID 0x0005) and all mandatory features associated with it
GAP ([Vol. 3] Part C)	All mandatory features in sections 9-12
ATT ([Vol. 3] Part F)	All mandatory features
GATT ([Vol. 3] Part G)	All mandatory features
SM ([Vol. 3] Part H)	All mandatory features

Table 4.6: LE Core Configuration Host requirements

Controller Part:

Layer	Required Features
PHY ([Vol. 6] Part A)	All mandatory features
LL ([Vol. 6] Part B)	All mandatory features

Table 4.7: LE Core Configuration Controller requirements

Tabla 2. Requisitos Bluetooth low energy. Fuente: Bluetooth Core specification 4.0

Ahora bien: ¿Cuáles son las características obligatorias? ¿Qué es el anfitrión y el controlador?, la respuesta a estas y otras preguntas se verá en las próximas páginas.

En primer lugar, como en el Bluetooth clásico, la pila de protocolos para el BLE tiene dos partes principales: El controlador (Controller) y el anfitrión (Host). El controlador comprende la capa física (Physical layer) y la capa de enlace (Link Layer) y suele implementarse normalmente en un SOC (system on chip) con radio integrada. El anfitrión se comunica con el controlador utilizando una interfaz estándar (HCI, Host Controller Interface) e incluye el protocolo de control y adaptación de enlace lógico (logical link control & adaptation protocol, L2CAP), encargado de controlar la comunicación proveniente de niveles superiores, el protocolo de atributos (Attribute Protocol, ATT), el protocolo de gestión de seguridad (Security Manager Protocol, SMP) o el perfil de acceso genérico (Generic Access Profile ,GAP), entre otros.

En la figura 2 y 3 se puede la estructura una pila de protocolo para BLE, incluyéndose dos versiones de la misma para un mayor soporte visual de los conceptos introducidos en las siguientes páginas.

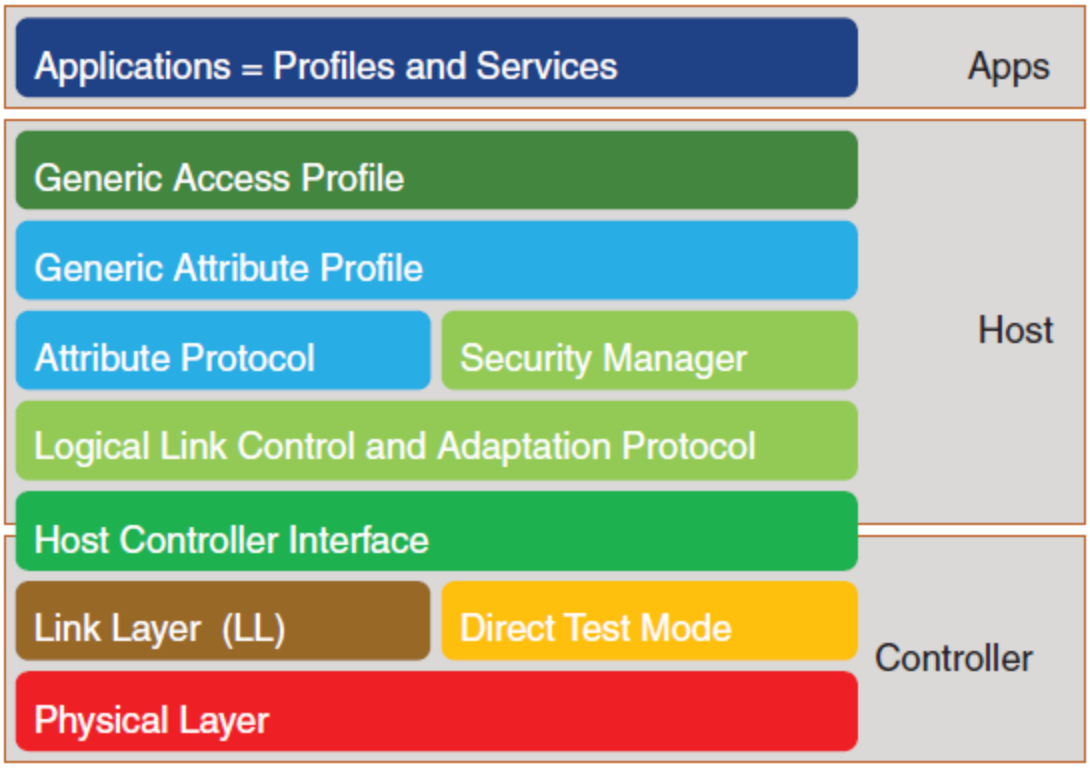
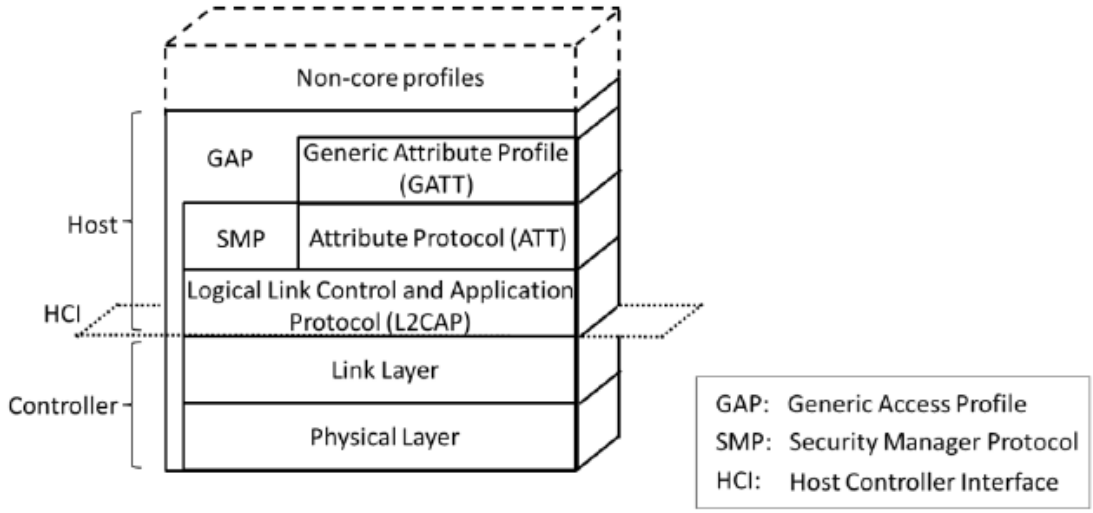


Figura 2 (arriba) y figura 3 (abajo). Pilas de protocolo para Bluetooth low energy. Fuente [10] [11]



Aunque algunas de las funcionalidades del controlador para BLE son heredadas del Bluetooth clásico, ambos tipos de controlador por lo general son incompatibles. Como se ha podido ver en la tabla 1 existen dos tipos de implementaciones, de modo único en las cuales solo se incluye la pila de protocolo de baja energía y de modo dual, donde se integra las funcionalidades de Bluetooth de bajo consumo en un controlador Bluetooth clásico.

2.2.2.1 Controlador Bluetooth

Capa física (*Physical Layer*)

La tecnología BLE fue diseñada para operar en la banda de radiofrecuencia ISM de los 2.4 GHz, legal en todo el mundo sin necesidad de licencia. A diferencia del Bluetooth clásico, el cual utiliza 79 canales físicos, BLE utiliza 40 canales espaciados entre si 2Mhz.

En BLE existen dos tipos de canales de radiofrecuencia, por un lado los canales utilizados para *advertising*, que son 3: El 37, 38 y 39. Estos canales no fueron escogidos al azar, ya que la elección de los mismos fue hecha con el fin de evitar los canales más comunes de Wi-Fi en la banda de 2.4 Ghz. Por otro lado están los canales para datos o *data channels*, los cuales conforman el resto de canales hasta completar los 40 que utiliza BLE. En la figura 4 se puede ver su distribución de manera clara.

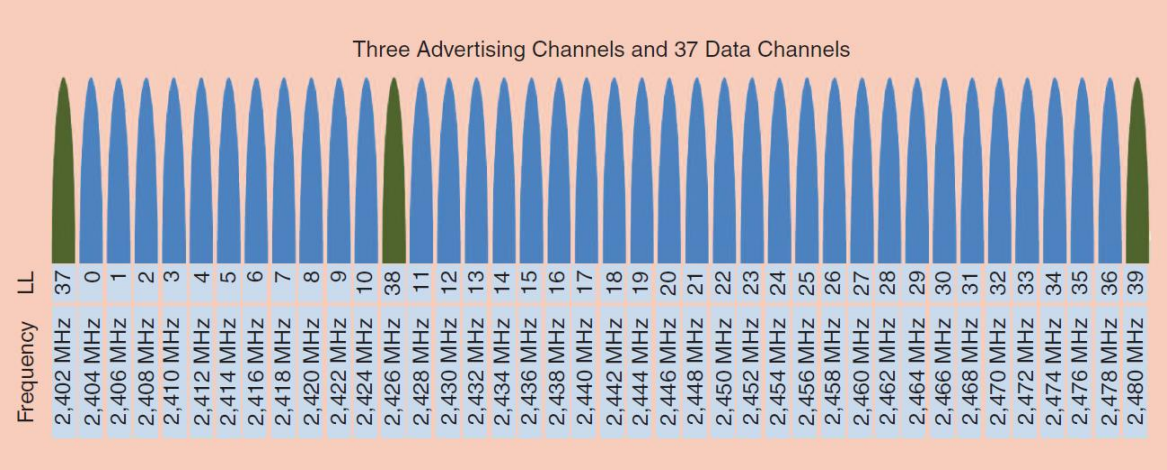


Figura 4. Canales utilizados en Bluetooth low energy. Fuente [10]

Los canales para *advertising* se utilizan para la detección de dispositivos cercanos, establecer conexiones y para realizar transmisiones, mientras que los canales de datos son usados para la comunicación bidireccional entre los dispositivos conectados.

Para operar en la banda ISM se utiliza un transceptor que ejecuta saltos de frecuencia (*frequency hopping*) en un conjunto amplio de portadoras. BLE utiliza, por tanto, al igual que Bluetooth clásico, un sistema de espectro de dispersión basado en saltos (*frequency hopping spread spectrum*), diseñado para evitar interferencias y empobrecimiento (fading) de la señal. Este mecanismo selecciona uno de los 37 canales de datos disponibles durante un intervalo de tiempo determinado. Además de ello, todos los canales físicos utilizan modulación GFSK (*Gaussian Frequency Shift Keying),* un tipo de modulación usada también en otras tecnologías inalámbricas como Z-Wave o Wavenis [12].

En la siguiente tabla (tabla 3) se pueden ver algunas de las características más relevantes a nivel físico de la tecnología BLE.

Table 2. Bluetooth Smart Fact Sheet.

Range	~150 m open field
Output power	~10 mW (10 dBm)
Max current	~15 mA
Latency	3 ms
Topology	Star
Connections	>2 billion
Modulation	GFSK @ 2.4 GHz
Robustness	Adaptive frequency hopping, 24-b CRC
Security	128-b AES CCM
Sleep current	~1 µA
Modes	Broadcast, connection, event data models reads, and writes

Note: items marked in blue are defined in the specification; other items are implementation specific.

Tabla 3. Hoja de características de Bluetooth low energy. Fuente [10]

Capa de enlace (*Link Layer*)

En BLE, cuando un dispositivo solo necesita transmitir datos, transmite esos datos en paquetes de *advertising* a través de los canales correspondientes. Cualquier dispositivo que transmita paquetes de a*dvertising* es conocido como un dispositivo *advertiser*. La transmisión de paquetes a través de los canales de *advertising* tiene lugar durante intervalos de tiempo llamados *advertising events*. Dentro de un *advertising event*, el *advertiser* usa secuencialmente cada canal de a*dvertising* para la transmisión de paquetes. Los dispositivos cuya única función es recibir datos a través de los canales de *advertising* son llamados *escaners*.

Debido a que BLE es asíncrono, las comunicaciones empiezan siempre con un *advertising packet*, que puede ocurrir en cualquier momento. Un dispositivo *advertiser*

enviará *advertising packets* cuando este buscando dispositivos a los que conectar, es decir estos paquetes anunciarán que el dispositivo está disponible. Por otro lado existe otro tipo de dispositivo, llamado cliente o *initiator,* el cual realiza la tarea opuesta al *advertiser*, es decir estará escuchando cuando pueda y responderá con un mensaje de petición de conexión al *advertiser* si quiere establecer una conexión, creando una conexión punto a punto entre ambos.

BLE define dos roles para los dispositivos en la capa de enlace durante una conexión, el rol de maestro (*master*) y el de esclavo (*slave*). Estos son los dispositivos que actúan como *initiator* y *advertiser* durante la creación de la conexión y que una vez conectados adoptan los papeles de maestro y esclavo respectivamente. Un maestro puede gestionar múltiples conexiones simultáneas con diferentes esclavos, mientras que un esclavo solo puede conectar a la vez con un solo maestro. La red formada por el maestro y sus esclavos es llamada *piconet* y seguirá una topología en estrella, con el maestro como nodo central. Un dispositivo BLE solo puede pertenecer a una *piconet* al mismo tiempo.

Para evitar un consumo excesivo los esclavos se encuentran por defecto en un modo de bajo consumo o *sleep mode* y se despiertan periódicamente para escuchar posibles recepciones de paquetes enviados por el maestro. El maestro determina cuando deben escuchar los esclavos, coordinando cuando accede cada uno mediante el uso de un sistema de multiplexación por división de tiempo o *Time Division Multiple Access* (TDMA). El maestro además provee al esclavo de la información necesaria para el algoritmo que controla los saltos de frecuencia (incluyendo los canales de datos que se van a utilizar) y para la supervisión de la conexión. Los parámetros relativos a la gestión de la conexión son transmitidos en un mensaje llamado *Connection Request* y pueden ser alterados durante la conexión debido a diferentes razones (p. ej. usar un nuevo mapa de canales de datos debido al cambio en el patrón de interferencias).

Una vez que la conexión entre el maestro y el esclavo ha sido creada, el canal físico se divide en unidades de tiempo no superpuestas llamadas *connection events*. Dentro de cada *connection event*, los paquetes son transmitidos por la misma frecuencia, es decir usando el mismo canal de datos. Cada conexión comienza con la transmisión de un

paquete por el maestro, si el esclavo recibe el paquete envía una respuesta al maestro confirmando que el paquete ha sido recibido. Por lo general, el maestro no enviará otro paquete hasta haber recibido la respuesta del esclavo y deberán transcurrir al menos 150 µs entre el final de la transmisión de un paquete y el envío de otro nuevo. El *connection event* se considerará abierto siempre y cuando el maestro y el esclavo continúen enviándose paquetes.

Los paquetes enviados a través de los canales de datos incluyen un *More Data* bit (MD) que señala cuando existe más información por transmitir por parte del que envía. Si ninguno de los dispositivos tiene más información que transmitir el *connection event* se cerrará y el esclavo no seguirá escuchando hasta el comienzo de un nuevo *connection event*. Otras circunstancias que pueden forzar que finalice un *connection event* pueden ser la recepción de dos paquetes consecutivos con errores por el maestro o el esclavo, o la corrupción del campo de la dirección de acceso de un paquete enviado a cualquier dispositivo. Además de ello, en cada paquete se incluye un CRC (código de comprobación de redundancia cíclica o *Cyclic Redundancy Check*) de 24 bits para detectar posibles errores en las comunicaciones.

Una vez finalizado el *connection event* anterior si se inicia uno nuevo el maestro y el esclavo utilizaran un nuevo canal de datos en una frecuencia diferente a la usada anteriormente calculada a partir del algoritmo que controla los saltos de frecuencia.

El tiempo entre el comienzo de dos *connection events* consecutivos es especificado mediante el parámetro *connInterval*, que es un múltiplo de 1.25 ms y cuyo valor suele encontrarse entre 7.5ms y 4s. Otro parámetro importante es el *connSlaveLatency,* el cual define el número de *connection events* consecutivos durante los cuales el slave no escuchará al maestro y por tanto podrá tener apagada la radio, disminuyendo el consumo derivado de esta. Este parámetro será un número entero entre 0 y 499.

Si el tiempo entre el último paquete recibido supera el parámetro *connSupervisionTimeout,* que se encuentra entre 100 ms y 32s se producirá un *supervision timeout*. El propósito de este mecanismo es detectar la perdida de conexión debida a interferencias o al movimiento fuera de rango de uno de los dispositivos involucrados en la transmisión.

Además de los diversos parámetros que ayudan a controlar el flujo de las comunicaciones, cada paquete enviado por el canal para datos contiene en su cabecera dos campos de 1 bit llamados *Sequence Number* (SN) y *Next Expected Sequence Number* (NESN). El SN identifica el paquete, mientras que el NESN indica que paquete deberá recibirse después. Si un dispositivo recibe un paquete, el NESN del siguiente paquete se verá incrementado y su recepción servirá como prueba de que el sistema está funcionando según lo previsto.

Advertising

Anteriormente se ha hablado de canales para *advertising*, paquetes de *advertising*, *advertising events*, etc. Pero, ¿cómo funciona el *advertising* en BLE?

A diferencia del Bluetooth clásico, para BLE no se necesita un proceso largo para encontrar dispositivos cercanos. En lugar de ello los dispositivos periféricos pueden transmitir en uno de los tres canales para *advertising* (a modo de recordatorio eran el 37, 38 y 39) enviando un paquete denominado *ADV_IND*. Estas transmisiones por paquetes son muy cortas, con solo 31 bytes disponibles para la carga útil, a los que hay que añadir los necesarios para la cabecera, la dirección MAC y los campos para el cheksum.

Mediante los paquetes *ADV_IND* también se puede transmitir información como por ejemplo el valor de un sensor una vez por segundo. Además, si 31 bytes no fueran suficientes para determinadas aplicaciones, existe al menos otra manera que permite mayor flexibilidad en este aspecto. Esta consiste en un mecanismo simple que permite un escaneo activo por parte del dispositivo central. Si un paquete *ADV_IND* es transmitido desde el dispositivo periférico indicando que soporta este tipo de escaneo, el dispositivo central entiende que están disponibles más datos, y por tanto transmite un paquete denominado *SCAN_REQ*. Como respuesta a este paquete, el dispositivo periférico devolverá un paquete denominado *SCAN_RSP* que contendrá otros 31 bytes de información útil. En la figura 5 se puede ver de manera gráfica este procedimiento.

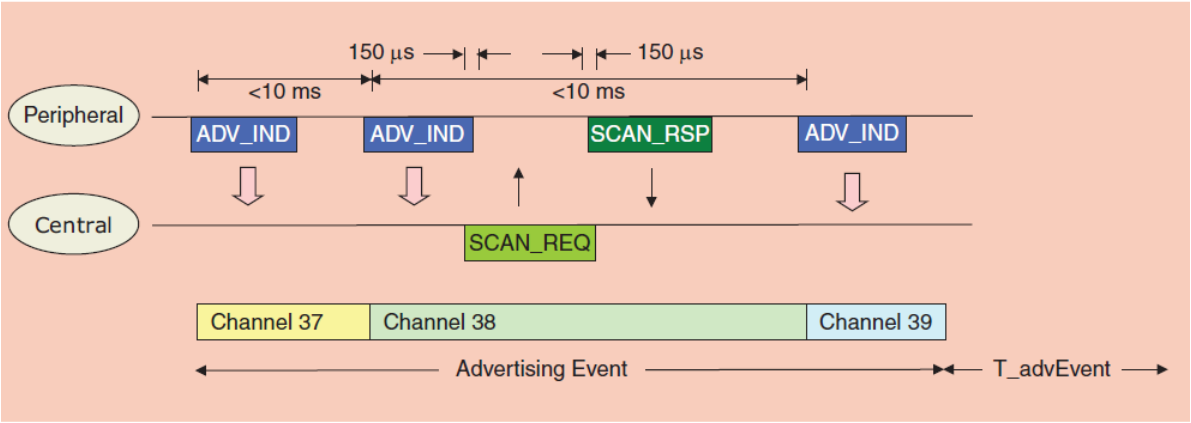


Figura 5. Transmisión de información durante *advertising events.* Fuente [13]

Sin embargo, aunque utilizar los paquetes de *advertising* para enviar información relevante pueda parecer una solución plausible para algunas aplicaciones, por lo general es preferible una transferencia de datos bidireccional y sin restricciones.

2.2.2.2 Anfitrión Bluetooth

Como pudo observarse en las figura 2 y 3, el *host* o anfitrión en la pila de protocolo para Bluetooth low energy incluye:

Protocolo de control y adaptación de enlace lógico (L2CAP)

El protocolo de control y adaptación de enlace lógico o L2CAP (*Logical link control and adaptation protocol*) utilizado en BLE difiere frente al utilizado en Bluetooth clásico. Es un protocolo diferente, más simplificado y optimizado. En BLE, la meta principal del L2CAP es multiplexar la información proveniente de tres niveles superiores, en concreto de ATT, SMP y del *control signaling* de la capa de enlace. La información de estos servicios es manejada por el L2CAP sin el uso de mecanismos de control del flujo ni retransmisiones, las cuales si están disponibles en otras versiones de Bluetooth. Tampoco se utilizan procesos de segmentación y re ensamblado de la información, ya que los protocolos de las capas superiores ya proporcionan la misma en un formato que no supera el máximo permitido que equivale a 23 bytes para BLE.

ATT *(Attribute protocol)*

El ATT define la comunicación entre dos dispositivos (cliente y servidor). El servidor necesita disponer de una serie de atributos, cada uno de los cuales está formado por

una estructura de datos que guarda la información gestionada por el GATT, que es el protocolo que opera por encima del ATT. El rol de cliente o servidor es determinado por el GATT, y es independiente del rol de maestro o esclavo mencionado con anterioridad. El cliente puede acceder a los atributos del servidor enviando peticiones, las cuales activaran mensajes de respuesta del servidor. Para un mejor resultado final, el servidor además puede enviar al cliente dos tipos de mensajes no solicitados que contienen atributos: por un lado notificaciones, que no requieren de confirmación, y por otro indicaciones, que requieren una confirmación por parte del cliente. Un cliente puede también enviar órdenes al servidor para realizar la escritura de los valores de los atributos.

GATT (*Generic Attribute profile*)

El GATT utiliza el ATT para el descubrimiento de servicios y el intercambio de características de un dispositivo a otro. Una característica es un conjunto de datos que incluye un valor y unas propiedades. La información relacionada con servicios y características es guardada en atributos. En el sistema diseñado en este proyecto, por ejemplo, un servidor que tiene un servicio llamado “sensor de temperatura” tendrá una característica llamada “temperatura” que usara un atributo para describir el sensor, otro atributo para guardar las medidas realizadas por el sensor y otro atributo para guardar las unidades en la que se están realizando las medidas.

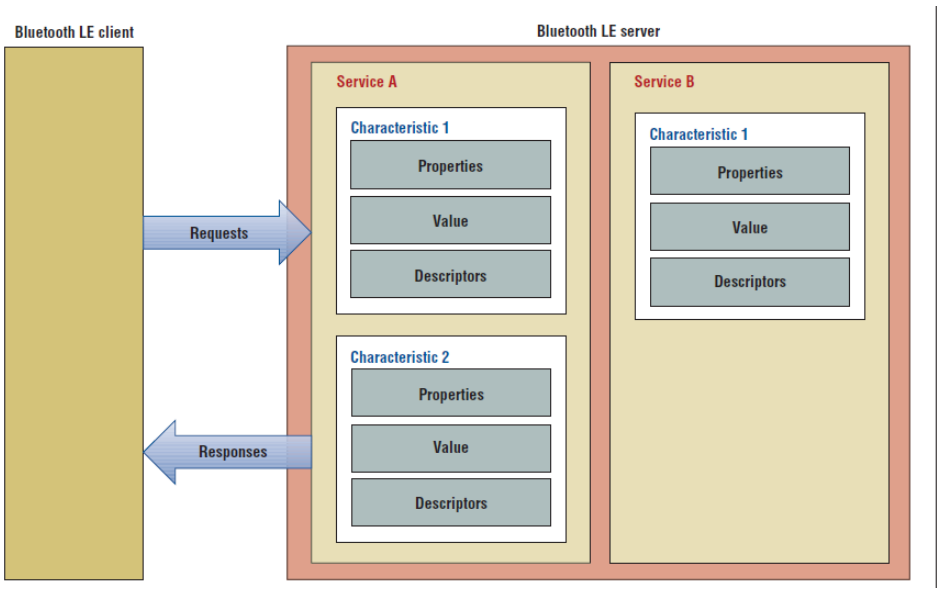


Figura 6. Protocolo ATT para Bluetooth low energy. Fuente [13]

GAP (*Generic Access Profile*)

En el más alto nivel de la pila de protocolos BLE se encuentra el GAP (Generic Access Profile) o perfil de acceso genérico. Es la base para todos los demás perfiles. El GAP define 4 roles, cada uno con diferentes características sobre el controlador que se encuentra por debajo: *broadcaster*, *observer*, periférico y central. Un dispositivo con el rol de *broadcaster* solo podrá transmitir información (a través de los canales de *advertising*) y no permitirá conexiones con otros dispositivos. El rol de *observer* es complementario al de *broadcaster*, es decir su objetivo es recibir la información transmitida por este último y nada más. Por otro lado los dispositivos centrales son los que se encuentran a cargo de iniciar y gestionar conexiones múltiples, mientras que los dispositivos que utilizan el rol periférico son aquellos que simplemente conectan con un dispositivo central. Como consecuencia de ello, los roles central y periférico requieren que el controlador del dispositivo soporte trabajar tanto en modo maestro como en modo esclavo. Un dispositivo puede soportar varios roles, pero solo uno estará activo cada vez.

Además de todos los perfiles descritos, la pita de protocolo de BLE permite perfiles adicionales construidos por encima del GAP. Como la tecnología Bluetooth utiliza una estructura jerarquizada de perfiles, es posible construir nuevos perfiles que incluyan todos los requisitos de un perfil ya existente. Un perfil de más alto nivel que especifica como las aplicaciones pueden operar entre si se denomina *Application profile* o perfil de aplicación.

La seguridad en BLE

Como es natural, la información transmitida debe estar protegida de algún modo ante ataques u otro tipo de riesgos que afecten el correcto funcionamiento del sistema. La tecnología BLE dispone de varios servicios de seguridad para proteger la información intercambiada entre dos dispositivos conectados entre sí. La mayoría de los servicios de seguridad disponibles pueden ser clasificados en términos de dos modos de seguridad llamados *LE Security Mode 1* y *LE Security Mode 2*. Estos dos modos proporcionan funcionalidades de seguridad en la capa de enlace y en la capa de ATT respectivamente.

La capa de enlace soporta tanto la encriptación como mecanismos de autenticación mediante el uso de códigos CCM (*Cipher Block Chaining-Message Authentication Code*) y cifrados de bloques mediante claves AES de 128 bits. Cuando se utilizan ambos para una conexión, se añade un código MIC (*Message Integrity Check*) de 4 bytes a la carga útil de PDU (*protocol data unit*). También es posible transmitir información autentificada sobre una conexión de capa de enlace sin encriptar. En este caso, una firma de 12 bytes es ubicada después de la carga útil en la capa ATT.

Además de los servicios de autenticación y cifrado de datos en la capa de enlace, la tecnología BLE también soporta un mecanismo llamado función de privacidad, que permite al dispositivo utilizar direcciones privadas y cambiarlas con frecuencia. La función de privacidad disminuye la posibilidad de que alguien puede rastrear un dispositivo que utilice BLE.

Antes se ha hablado de que existían dos modos de seguridad bajo los cuales podían clasificarse la mayoría de servicios de seguridad disponibles. En cada uno de estos modos pueden encontrarse diferentes niveles, cuyas características se pueden ver en la tabla 4.

		Pairing	Encryption	Data Integrity	Layer
LE Security Mode 1	**Level 1**	No	No	No	Link Layer
	Level 2	Unauthenticated	Yes	Yes	
	Level 3	Authenticated	Yes	Yes	
LE Security Mode 2	**Level 1**	Unauthenticated	No	Yes	ATT layer
	Level 2	Authenticated	No	Yes	

Tabla 4. Características de cada nivel de los modos de seguridad en BLE. Fuente [11]

En esta tabla se observa que cada modo de seguridad cuenta con varios niveles, y que según el nivel que se escoja el *pairing* es necesario o no, y precisa de realizarse con autenticación o no, pero ¿cómo se realiza el *pairing*?

En primer lugar todas las comunicaciones relativas al proceso de *pairing* son realizadas por el protocolo SMP o *Security Manager Protocol*, este es el que define como realizar un enlace seguro así como gestiona el intercambio de claves. El *pairing* comprende 3 fases, en la primera, los dos dispositivos conectados anuncian sus capacidades de entrada/salida de datos y, basándose en ellas, se elige un modelo de asociación entre

los disponibles para realizar el *pairing* (*Out of Band*, *Passkey Entry* o *Just Works*). El primer modelo, denominado modelo *Out of Band,* utiliza un método externo (p. ej. NFC) para intercambiar información necesaria para realizar el *pairing*, el cual se completa después utilizando esa información mediante los canales usuales de Bluetooth. El método externo utilizado debe resistir ataques *man-in-the-middle* o MITM.

El segundo modelo, denominado *Passkey Entry*, se utiliza cuando uno de los dos dispositivos no tiene la capacidad de mostrar 6 dígitos pero si la de aceptarlos cuando sean introducidos en él y el otro si puede mostrarlos. Así, este último mostrará los dígitos y le pedirá al usuario que los introduzca en el otro dispositivo. Si el valor introducido es el correcto el *pairing* se hará efectivo.

Por último el tercer modelo, denominado *Just Works,* simplemente funciona como indica el propio nombre. Este método no requiere interacción por parte del usuario pero no proporciona protección contra ataques de tipo *man-in-the-middle.*

Una vez elegido el modelo la segunda fase tiene el objetivo de generar una clave denominada *Short-Term Key* (STK), la cual será utilizada en la tercera fase. Esta clave se generara basándose en una clave temporal o *Temporary Key* (TK) escogida por los dispositivos anteriormente mediante el modelo elegido, dando lugar al final de la segunda fase.

En la tercera fase, cada *endpoint* de la conexión puede distribuir al otro *endpoint* hasta 3 claves de 128 bits, las cuales son: *Long-Term Key* (LTK), *Connection Signature Resolving Key* (CSRK e *Identity Resolving Key* (IRK). La clave LTK se usa para generar una clave de 128 bits utilizada para la encriptación y autenticación de la capa de enlace. La clave CSRK se utilizara en la capa ATT y la clave IRK servirá para generar una dirección privada basada en la dirección pública del dispositivo. La clave STK generada en la segunda fase servirá para encriptar las comunicaciones requeridas para el intercambio de estas tres claves.

A pesar de todas las medidas de seguridad que existen actualmente para las comunicaciones realizadas mediante la tecnología BLE, existen aún puntos débiles o

fallos que hacen que el sistema sea vulnerable. Uno de estos puntos débiles es que los métodos utilizados en el *pairing* para BLE no protegen contra el llamado *passive eavesdropping*, que traducido de manera literal sería algo así como escuchas ilegales. Mediante este método el atacante puede obtener los mensajes que contienen las claves intercambiadas en el proceso y a partir de ahí ser capaz de "escuchar" todo lo que viene después. Mientras que en el Bluetooth clásico no existe esta vulnerabilidad ya que las claves son encriptadas mediante el protocolo ECDH (Elliptic Curve Diffie Hellman), en el BLE aún no existe protección ante este tipo de ataques.

2.2.3 Aplicaciones

Debido a la fuerte presencia de la que dispone la tecnología Bluetooth en el mercado de los dispositivos móviles, y dado que BLE y clásico pueden integrarse fácilmente en el mismo dispositivo simplemente utilizando un chip que permita el modo dual, no es de extrañar que en los próximos años se vea un aumento considerable del número de dispositivos que utilicen esta tecnología, así como de las aplicaciones que utilicen BLE para realizar sus comunicaciones. Es lógico pensar que, mientras la tecnología BLE aún está empezando a implantarse, otras tecnologías orientadas a aplicaciones de bajo consumo como ZigBee, 6LoWPAN o Z-Wave ya cuentan con una representación numerosa en algunos segmentos del mercado. Sin embargo, a largo plazo la ventaja de la que dispone la tecnología BLE en el mercado de dispositivos móviles (p.ej. smartphones) será un factor que posiblemente incline la balanza en favor de esta última.

Una muestra de las aplicaciones que pueden verse beneficiadas por el uso de la tecnología BLE o ya la utilizan en la actualidad es:

- Sistemas de control y monitorización ambiental [14]: Sistemas de calefacción, ventilación, aire acondicionado, control de humedad, luces, etc.

- Sistemas de seguridad: Desde sensores de movimiento hasta sensores de apertura en ventanas o puertas[15], pasando por almacenamiento seguro de documentos [16], las aplicaciones de seguridad tienen aún mucho potencial en el campo de los dispositivos móviles.

- Sistemas de monitorización de la salud: Presión arterial, ritmo cardíaco, nivel de oxígeno en sangre,… Como ejemplo de este tipo de aplicación se encuentra el sistema diseñado en [17], el cual utiliza la tecnología BLE para transmitir los datos obtenidos del electrocardiograma.

- Monitorización de la actividad física: El cuidado de la salud, bienestar y deportes constituyen un campo en el cual el uso de Bluetooth [18][19] no es algo nuevo y para el cual la utilización de la tecnología BLE constituye un avance significativo [20]. De hecho, la *Continua Health Alliance*, organización que agrupa a numerosas tanto en el ámbito del cuidado de la salud como tecnológicas [21], ha anunciado la selección de la tecnología BLE para las aplicaciones que tengan como objetivo monitorizar la actividad así como la frecuencia cardiaca de una persona [22]. Como producto ya disponible en el mercado que utiliza la tecnología BLE se tiene por ejemplo, la pulsera Flex de la compañía americana Fitbit [23].

- Detección de proximidad: La localización de dispositivos y el envío de alertas cuando estos se encuentran fuera de rango constituye una de las aplicaciones más prolíficas para BLE. Mediante el uso de pequeñas etiquetas ubicadas sobre cualquier aparato u objeto, como por ejemplo llaves, ordenadores, libros, bolsas, es posible localizar los mismos mediante BLE y un dispositivo móvil. Existe ya múltiples productos en este ámbito que utilizan BLE como por ejemplo Proximo [24], Tile [25] y StickNFind [26].

- Extensión de interfaces de usuario: Aplicaciones tales como mostrar alarmas del smarthphone en el reloj [27] o la pulsera inteligentes, etc.

- Métodos inalámbricos de pago: A pesar de que ya existen tecnologías aplicables para este tipo de aplicaciones como pueda ser la tecnología de Comunicación de Campo Cercano o *Near Field Communication* (NFC) [28], esta ofrece muy poca flexibilidad en cuanto a la localización del dispositivo frente a la tecnología BLE, teniendo que encontrarse a corta distancia para funcionar.

3 Diseño de la solución técnica

En este capítulo se expone el diseño del sistema, explicando cuáles son sus componentes y las diferentes características del mismo. En primer lugar se verá el conjunto de forma global mediante el diagrama de bloques del sistema, con una breve explicación de la funcionalidad esperada de cada elemento. A continuación, se analizará cada componente por separado, profundizando en su funcionamiento así como en la relación del mismo con el resto de componentes del sistema.

3.1 Diagrama de bloques del sistema

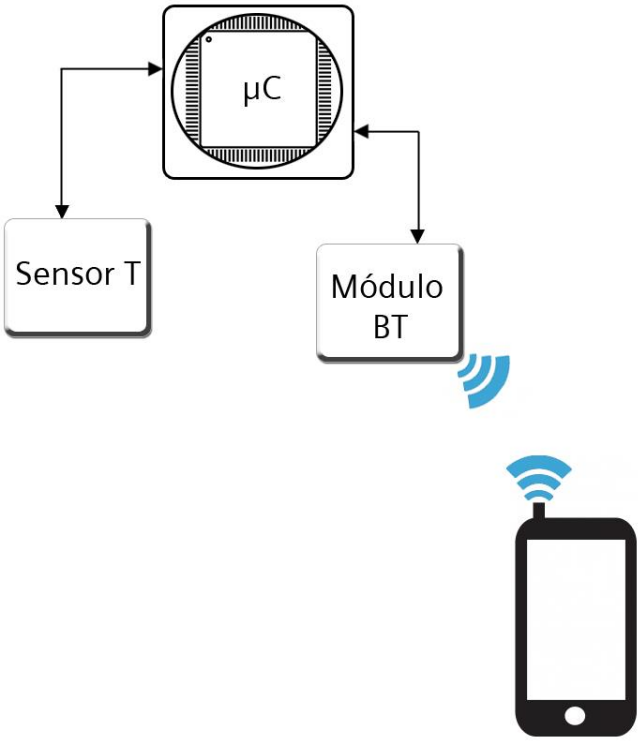


Figura 7. Diagrama de bloques del sistema.

Como se puede observar en la figura 7 el microcontrolador se ubica como pieza central del sistema, siendo el encargado de coordinar y marcar el ritmo de funcionamiento del mismo, controlando cuando y como se realiza cada tarea.

3.2 Requerimientos básicos del sistema

- El sensor de temperatura y el modulo Bluetooth actuarán siempre como esclavos (slaves) del microcontrolador.
- Debido a las restricciones impuestas por el hardware utilizado, ambos se comunicarán mediantes bus de comunicación en serie con el microcontrolador, utilizando I^2C para las comunicaciones entre sensor de temperatura y microcontrolador y SPI para las comunicaciones entre microcontrolador y modulo Bluetooth.
- El modulo Bluetooth además de comunicar con el microcontrolador mediante SPI podrá comunicar con un dispositivo móvil mediante Bluetooth low energy.
- El sistema completo deberá tener unas dimensiones máximas de 10 x 5 x 3 cm (largo x ancho x alto).
- Mediante el dispositivo móvil se podrá como mínimo visualizar en pantalla el valor medido de temperatura, generar gráficas que muestren la evolución de la misma a lo largo del tiempo, activar alarmas al superar unas temperaturas máximas o mínimas establecidas por el usuario y permitir notificaciones vía correo electrónico y/o SMS.

3.3 Elección de los componentes hardware del sistema

Conociendo las características básicas a las que se debe ajustar el sistema, es hora de elegir los componentes hardware que permitan implementarlas. Se procederá pues a escoger los componentes atendiendo tanto a criterios técnicos como a aspectos físicos (p.ej. tamaño), precio, etc.

3.3.1 Sensor de temperatura

La variable elegida a medir por el sistema ha sido la temperatura, si bien como se ha explicado en la introducción el sistema está diseñado de forma modular, facilitando el cambio de esta parte del sistema al no estar el mismo incorporado al microcontrolador o al módulo Bluetooth.

Aunque existen en el mercado soluciones que incorporan un sensor de temperatura integrado en la propia placa, (p. ej. el módulo Bluetooth elegido incorpora un sensor de temperatura) se optó por un diseño orientado a permitir cierta libertad sobre la distancia a la que se ubica el sensor (dentro de las restricciones del bus de comunicaciones, ya que existen límites en esta distancia que si se sobrepasan pueden afectar negativamente a las comunicaciones (perdida de datos en la comunicación) e incluso inhabilitarlas por completo) y a la posible ampliación del sistema, teniendo en mente posibles modificaciones como pueda ser el incremento en el número de sensores utilizados, ubicando cada uno en una posición diferente y funcionando de manera simultánea.

A la hora de elegir el sensor de temperatura en primer lugar hay que decidir si este va a ser digital o analógico. Las ventajas del primero frente al segundo son varias. Los sensores digitales permiten comunicaciones directas con el microcontrolador, sin necesidad de conversores y además no precisan de circuitos electrónicos adicionales. Esto se traduce en mayores facilidades a la hora de realizar el montaje y en un ahorro de materiales, lo cual incide directamente en el costo final del proyecto.

El siguiente paso es decidir cuál de las alternativas en sensores de temperatura digitales que ofrece el mercado se ajusta más a las especificaciones del sistema y al diseño del mismo. Desde el principio se tuvo en cuenta el precio por unidad de cada sensor y las prestaciones del mismo. Así mismo el sensor elegido debía poder comunicarse en serie con el microcontrolador y presentar un tamaño reducido. El TC74 fabricado por Microchip y el ADT7420 fabricado por Analog Devices fueron dos de los sensores elegidos que más se ajustaban a las necesidades del sistema, siendo elegido finalmente el TC74. Aunque el ADT7420 es superior en cuanto a precisión (±0.25°C en condiciones normales frente a los ±2°C del TC74) y resolución (0.0078 °C frente a 1 °C

del TC74) se refiere, su mayor consumo (210µA frente a los 200µA del TC74 en modo normal a 5.5v) y sobre todo, su mayor coste (entre 5 y 10 veces más caro que el TC74 según proveedor), declinó la balanza a favor del TC74.

El TC74 permite comunicaciones en serie mediante bus I²C, puede funcionar en modo de bajo consumo y funciona en el rango de los 3 a los 5.5V, existiendo dos versiones del mismo, una de 3.3V y otra de 5V. Este último dato es especialmente relevante ya que puede acarrear circuitería adicional para conectar el sensor al sistema.

En este caso esto no será necesario ya que el sensor será alimentado por el microcontrolador a 5V, por tanto la versión de 5V del sensor de temperatura será la elegida. En el capítulo 4 **"Implementación del sistema"** se puede ver en detalle cómo se realizan las conexiones con el microcontrolador y como se realizan las comunicaciones entre el mismo y el sensor.

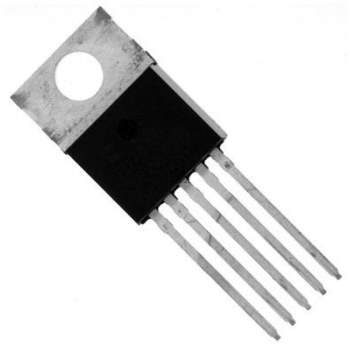


Figura 8. Sensor de temperatura TC74.

3.3.2 Módulo BLE

Las comunicaciones entre el sistema y el dispositivo móvil se realizarán mediante BLE, por lo cual será necesario escoger un chip que soporte ese tipo de comunicaciones. En este caso el chip elegido será el nRF8001 fabricado por Nordic Semiconductor. Debido a que será necesario comunicar el chip con el microcontrolador se utilizará el módulo nRF2740 del mismo fabricante, que trae incorporado el nRF8001 además de una antena PCB (*printed circuit board*) y 10 pines, 7 de los cuales serán utilizados para las comunicaciones con el microcontrolador. Existe otro módulo, el nRF2741, que

sustituye la antena PCB por un conector SMA posibilitando la utilización de una antena externa en el caso de necesitar un alcance mayor. En la figura 9 se puede observar el módulo nRF2740 así como una leyenda con algunos de sus componentes más importantes.

3-Conector P1 (10 pines)

4- nRF8001

5- Antena PCB

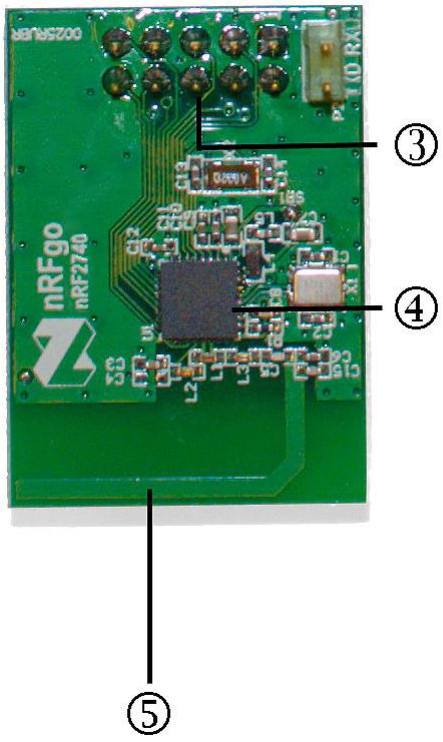


Figura 9. Módulo nRF2740.

El nRF8001 soporta comunicaciones en serie mediante bus SPI e integra una radio compatible con BLE a nivel de capa física (PHY), controlador de enlace en modo esclavo y anfitrión además de otras funcionalidades adicionales como un sistema de monitorización de batería o un sensor de temperatura. Puede ser alimentado en el rango de 1.6 a 3.6V, siendo 3.0V el valor nominal.

3.3.3 Microcontrolador

Existen múltiples alternativas a la hora de elegir un microcontrolador que se ajuste sin problemas a los requisitos técnicos del diseño planteado, por ello el primer criterio ha sido filtrar por tamaño, descartando placas de desarrollo que superasen los 10 centímetros de largo por 5 de ancho ya que estas son las medidas máximas del sistema

completo. Esta restricción se debe a que el sistema de medición de temperatura diseñado debe ser manejable y lo más compacto posible, dentro de las restricciones de tamaño inherentes al propio diseño, como es la utilización de tres componentes (microcontrolador, sensor de temperatura y módulo Bluetooth) cuando podría utilizarse un solo componente que englobase todas las funciones requeridas. En su lugar se optó por un diseño orientado a permitir cierta libertad sobre la distancia a la que se ubican los componentes y por tanto el resto de decisiones han sido tomadas de acuerdo con ello.

En segundo lugar el microcontrolador debe soportar comunicaciones mediante SPI para comunicarse con el módulo BLE e I²C para comunicarse con el sensor de temperatura, aunque como se verá más adelante ambos protocolos pueden ser implementados mediante *bit banging*.

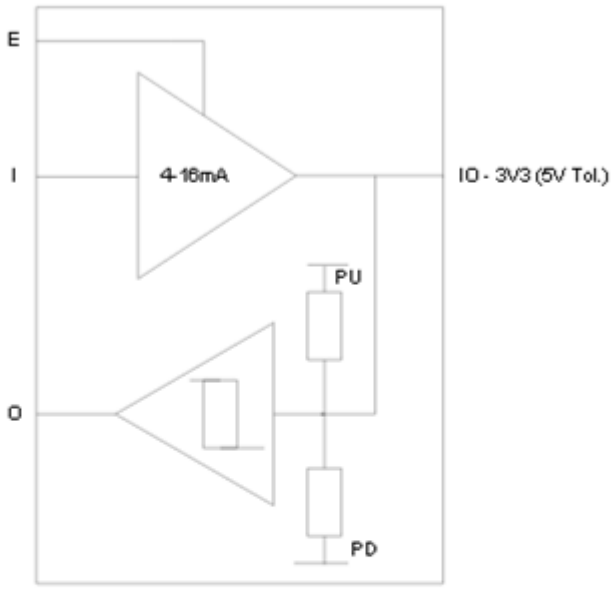
V2DIP1-32

Figura 10. V2DIP1-32.

La placa de desarrollo (*development board*) V2DIP1-32 (figura 10) fabricado por FTDI contiene un microcontrolador VNC2 de 16 bit del mismo fabricante, capaz de trabajar a una frecuencia de reloj máxima de 12MHz. Dispone de 256KBytes de memoria Flash y 16KBytes de RAM, interfaces para comunicaciones SPI y permite modos de funcionamiento en bajo consumo, todo ello incluido en un tamaño de 53.22 (largo) x 17.78 (ancho) mm. Existen varias versiones en función del número de pines disponibles, siendo 32 la opción más baja que permite el fabricante.

El V2DIP1-32 se alimenta a 5V, pudiendo ser alimentado desde el PC mediante USB o con una fuente de alimentación externa. Dispone de 12 IO pines (figura 11), en los cuales es posible controlar el nivel de corriente saliente en modo output (4mA, 8mA, 12mA y 16mA), así como el *slew rate*, el cual puede ser rápido o lento.

Además, resistencias de pull-up y pull-down de 75KΩ se incluyen dentro del micro, con el consiguiente ahorro de material y de circuitería externa que ello supone. Esta funcionalidad es útil en las comunicaciones mediante I^2C entre el sensor y el microcontrolador.



LVTTL	Low Voltage Transistor Transistor Logic
E	Enable to enable the output stage
I	Input
O	Output
IO	Input/Output
PU	Pull up resistor (75kOhm) option
PD	Pull down resistor (75kOhm) option

Figura 11. Configuración de un pin IO dentro del V2DIP1-32.

Módulo VNC2 Debugger / Programmer

Además del V2DIP1-32 es preciso adquirir un módulo adicional mediante el cual se realizará la programación en el microcontrolador y depuración de posibles errores derivados de la misma. Este módulo (figura 12) proporciona conectividad USB al microcontrolador, permitiendo acceder a las herramientas disponibles para el desarrollo de aplicaciones, como son las proporcionadas por el fabricante llamadas *Vinculum-II IDE development tools,* las cuales se instalaran en el PC. Este módulo no es necesario una vez programado el firmware de nuestra aplicación en el microcontrolador, siendo solo un añadido temporal que permite un sistema final de tamaño reducido.

Figura 12. Módulo VNC2 Debugger / Programmer.

Puntos a favor:

-Permite alimentar componentes externos a 3.3 o 5V.

-Soporte nativo para comunicaciones SPI.

-Permite alimentación externa (no USB).

-Permite hacer pull-up sin circuitería externa.

-IDE gratuito: Vinculum II IDE.

Puntos en contra:

-Necesita un módulo adicional para realizar la programación.

Arduino Micro

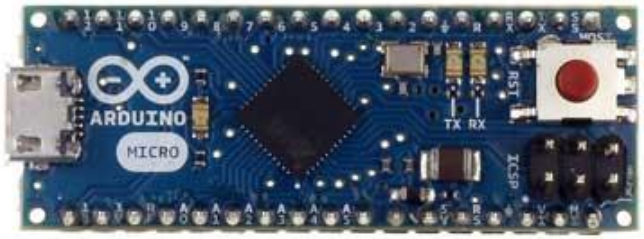


Figura 13. Arduino Micro.

La placa de desarrollo elegida: Arduino Micro (figura 13). Esta placa de desarrollo contiene un microcontrolador de 8 bits basado en el ATmega32u4 de la familia AVR y fabricado por Atmel, pudiendo trabajar a una frecuencia de reloj máxima de 16MHz. Dispone de 20 IO pines cada uno de los cuales puede proporcionar hasta 40 mA con resistencias internas disponibles de pull-up de 20-50 kOhms. Además, contiene 32KBytes de memoria Flash, 2.5KBytes de memoria SRAM y 1KBytes de memoria EEPROM. Dispone de soporte nativo para comunicaciones SPI (Master/Slave) e I^2C, permite interrupciones internas y externas y seis modos diferentes de bajo consumo. Las medidas del Arduino Micro son de 48 (largo) x 17.7 (ancho) mm.

El Arduino Micro puede ser alimentado vía USB o con una fuente de alimentación externa (6-20V) y no requiere de ningún modulo adicional para ser programado.

Puntos a favor:

-Es la alternativa con medidas más reducidas.

-Soporte nativo para comunicaciones SPI e I^2C tanto en hardware como en software mediante librerías.

-Las librerías incluidas en el SDK de Nordic para el chip Bluetooh low energy utilizado, el nRF8001, han sido probadas en dispositivos Arduino.

- Permite alimentar componentes externos a 3.3 o 5V.

- Permite alimentación externa (no USB).

-Permite hacer pull-up sin circuitería externa.

-Tamaño más reducido en comparación con el V2DIP1-32.

-Precio inferior al V2DIP1-32 (incluyendo el Módulo VNC2 Debugger / Programmer).

-IDE gratuito: Arduino IDE.

MINI-M4

Figura 14. MINI-M4 para STM32

La placa de desarrollo MINI-M4 (figura 14) del fabricante MikroElektronika contiene un microcontrolador STM32F415RG de 32bits de la familia STM32 fabricado por STMicroelectronics capaz de funcionar a una frecuencia de reloj máxima de 16MHz. Dispone de soporte nativo para comunicaciones SPI (Master/Slave) e I²C, permite interrupciones internas y externas y diferentes modos de bajo consumo (Sleep, Stop y Standby). Además, dispone de 18 IO pines capaz de ser configurados en modo pull-up con resistencias de 30-50 kOhms. Contiene 1 Mbyte de memoria FLASH, 192+4 Kbytes de memoria SRAM y un sensor de temperatura integrado. Las medidas de la MINI-M4 son de 50.8 (largo) x 17.78 (ancho) mm.

La MINI-M4 puede ser alimentada vía USB o con una fuente de alimentación externa y no requiere de ningún modulo adicional para ser programada.

Puntos a favor:

-Soporte nativo para comunicaciones SPI e I²C.

-Permite alimentación externa (no USB).

-Permite hacer pull-up sin circuitería externa.

-IDE gratuito: Al utilizar un microcontrolador STM se puede utilizar una amplia variedad de IDE: ATOLLIC, CooCox, HITEX, IAR, KEIL, RAISONANCE,…

Puntos en contra:

-No permite alimentar componentes externos a 5V, encareciendo la posible ampliación del sistema con periféricos que requieran ese voltaje.

Razones por las que se ha escogido la opción Arduino Micro

A pesar de que las tres alternativas presentadas tienen características similares, cumpliendo todas ellas los requerimientos básicos del sistema, esta placa tiene algunas características extra que hacen que la balanza se incline a su favor. En primer lugar es la placa que mayor facilidades otorga al usuario para realizar las comunicaciones vía SPI e I²C, con librerías precargadas que agilizan la programación. No es la alternativa más potente, siendo la MINI-M4 de 32 bits la ganadora en este campo, pero para los requerimientos del sistema, la capacidad de la Arduino Micro es más que suficiente. Por otro lado es la opción más económica y la que presenta un tamaño más reducido, motivos que consolidan a la Arduino Micro como la opción elegida para implementar el sistema.

3.3.4 Dispositivo móvil

Pese a que no entra en el diseño planteado el definir marca, modelo, u otras características del dispositivo móvil que vaya a ser utilizado por el usuario para interactuar con el sistema (mediante el software creado), si conviene puntualizar que existen ciertas restricciones tanto en hardware como en software para el mismo. Estas restricciones son:

- Debe ser compatible con BLE (Bluetooth Smart o Smart Ready)
- Debe disponer de la versión 4.3 (API nivel 18) o superior del sistema operativo Android. Esto es debido a que esta versión ha sido la que ha introducido un soporte de plataforma ya integrado para BLE. En el apartado 3.4.3 se explica el porqué de la elección del sistema operativo Android sobre el resto.

En este caso el dispositivo móvil utilizado para desarrollar y probar la aplicación Android ha sido una tableta Google Nexus 7 en su versión compatible con Bluetooth 4.0.

3.4. Aplicación para dispositivos móviles

A continuación se hace una descripción detallada del diseño de la aplicación. Posteriormente, se incluirá un diagrama de casos (figura 15) y se analizarán los distintos sistemas operativos móviles que existen en la actualidad, definiendo en cuál de ellos se desarrollará la aplicación.

3.4.1 Diseño de la aplicación móvil

El diseño de la aplicación determina cuales son las funcionalidades que debe tener y distingue dos categorías dentro de las mismas: Principales y secundarias.

Funciones principales

- Mostrar al usuario una interfaz visual que incluya la temperatura en tiempo real. Esta interfaz debe mostrar la temperatura en grados Celsius.

- Incluir una marca horaria que muestre la fecha y hora de la última toma de temperatura y detalles relativos al sensor de temperatura utilizado.
- Permitir la activación y desactivación de las comunicaciones vía BLE.
- Permitir enviar notificaciones vía correo electrónico y/o SMS.
- Permitir notificaciones mediante sonidos y/o alarmas.
- Configuración de las condiciones de alarma.

Funciones secundarias

- Mostrar la potencia con la que se recibe la señal Bluetooth por parte del dispositivo móvil.
- Mostrar gráficamente el valor de la temperatura medido a lo largo del tiempo.
- Almacenar los datos recopilados del sensor en un archivo alojado en la memoria interna del dispositivo móvil.

3.4.2. Diagrama de casos de uso de la aplicación móvil

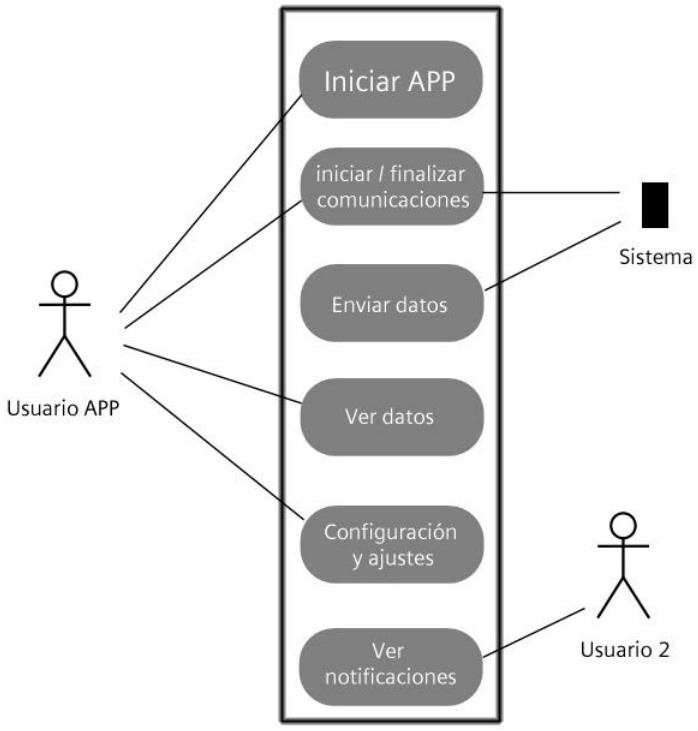


Figura 15. Diagrama de casos de uso de la aplicación.

3.4.3. Elección del sistema operativo móvil

A la hora de diseñar la aplicación móvil es fundamental elegir para qué plataforma o sistema operativo se va a desarrollar, ya que esta elección condicionará el resto del proceso debido a las singularidades que presenta cada sistema operativo y a las diferentes herramientas necesarias para su desarrollo.

Sistemas operativos móviles en la actualidad

En la actualidad existen numerosos sistemas operativos móviles, destacando entre ellos los siguientes:

- **Android:** El sistema operativo Android es sin duda el líder del mercado móvil en sistemas operativos, como puede verse en la figura 16. Diseñado originalmente para cámaras fotográficas profesionales, fue vendido a Google y modificado para ser utilizado en dispositivos móviles con pantalla táctil, como teléfonos inteligentes o tabletas, y también en relojes inteligentes, televisores y automóviles. Anunciado en 2007 y liberado en 2008, Android está basado en el kernel de Linux. Es una plataforma de desarrollo libre de código abierto, a excepción del software propiedad de Google como Play Store, Google Search, Google Play Services o Google Music. La última versión estable es la 4.4.4, llamada KitKat, y en pruebas la versión Android L.

- **iOS:** Propiedad de la compañía Apple y originalmente desarrollado para el iPhone, su uso se ha extendido a otros dispositivos como el iPod Touch, iPad y el Apple TV sin permitir su instalación en hardware de terceros. Anteriormente denominado iPhone OS, este derivado de Mac OS X fue lanzado en 2007, siendo hasta la fecha software propietario. El 17 de octubre de 2007, Steve Jobs anunció que un Kit de desarrollo de software o SDK estaría disponible para terceros y desarrolladores en febrero del 2008. El SDK Fue liberado finalmente el 6 de marzo de 2008, permitiendo así a los desarrolladores hacer aplicaciones para el iPhone y iPod Touch, así como probarlas en el "iPhone simulator". La última versión estable es la 7.1.2 encontrándose en pruebas la versión 8.0.

- **Windows Phone:** Anteriormente llamado Windows Mobile, Windows Phone es un sistema operativo móvil compacto desarrollado por Microsoft de código cerrado.

Lanzado al mercado el 15 de febrero de 2010, este sistema operativo integra todos los servicios de Microsoft tales como OneDrive, Office, Xbox Music, Xbox Video, Xbox Live games y Bing. En septiembre de 2011 Microsoft pone a disposición de los desarrolladores y de manera gratuita el Kit de desarrollo de software Windows Phone SDK 7.1, el cual ofrece todas las herramientas que necesita para desarrollar aplicaciones y juegos para dispositivos con Windows Phone 7.0 y Windows Phone 7.5. La última actualización recibida por este sistema operativo ha sido la versión 8.1.

- **Blackberry:** BlackBerry OS es un sistema operativo móvil desarrollado por la compañia RIM para los dispositivos BlackBerry. Su desarrollo se remonta la aparición de los primeros handheld en 1999 aunque fue lanzado oficialmente en el año 2003. Los desarrolladores pueden crear programas para BlackBerry pero en el caso de querer tener acceso a ciertas funcionalidades restringidas estos necesitan ser firmados digitalmente y asociados a una cuenta de desarrollador de RIM. La última versión conocida es la 10.2.1.

- **Firefox OS:** Firefox OS es un sistema operativo móvil, basado en HTML5 con núcleo Linux, de código abierto para varias plataformas. Lanzado por Mozilla Corporation el 23 de abril de 2013 bajo el apoyo de otras empresas y una gran comunidad de voluntarios de todo el mundo. El sistema operativo está diseñado para permitir a las aplicaciones HTML5 comunicarse directamente con el hardware del dispositivo usando JavaScript y Open Web APIs. El 2 de julio de 2013 la compañia Telefónica comenzó la venta del primer terminal con Firefox OS, el ZTE Open. La última versión estable conocida es la 1.3, estando en pruebas la 1.4.

- **Sailfish O:** Sailfish OS (Sistema operativo Pez Vela) es un sistema operativo para teléfonos inteligentes que está siendo desarrollado por la empresa finlandesa Jolla Ltd. en base al código fuente del proyecto Mer, la versión 5 de la biblioteca de desarrollo Qt y el protocolo de comunicación para servidores de visualización Wayland. El 20 de mayo de 2013 salió a la luz el primer dispositivo que utiliza este sistema operativo, llamado Jolla. La mayor parte del sistema operativo es software de código abierto, siendo la última versión estable conocida del mismo la 1.0.5.19, llamada Paarlampi.

• **Symbian:** Symbian es un sistema operativo propiedad de Nokia, producto de la alianza de varias empresas de telefonía móvil: Nokia, Sony Mobile Communications, Psion, Samsung, Siemens, Arima, Benq, Fujitsu, Lenovo, LG, Motorola, Mitsubishi Electric, Panasonic y Sharp entre otras. Las aplicaciones compatibles con Symbian se desarrollan a partir de lenguajes de programación orientados a objetos como C++, Java (con sus variantes como PJava, J2ME, etc.), Visual Basic para dispositivos móviles, entre otros, incluyendo algunos lenguajes disponibles en versión libre. En la actualidad es un software descontinuado, datando su última versión del año 2012 (Symbian OS 10.0).

• **Tizen:** Tizen es un sistema operativo móvil basado en Linux lanzado el 5 de enero del 2012 y patrocinado por Linux Foundation y la Fundación LiMo. Las interfaces de desarrollo de Tizen están basadas en HTML5 y otros estándares web y será diseñado para su uso en tabletas, netbooks, teléfonos inteligentes, televisores inteligentes y sistemas integrados de información y entretenimiento. Aunque originalmente fue presentado como un sistema operativo de código abierto, Tizen ha complicado su modelo de licencias. Su SDK está construido sobre componentes de código abierto, pero el SDK completo ha sido publicado bajo una licencia de Samsung de código no abierto. Su última versión conocida es la 3.0 llamada Magnolia.

• **Ubuntu Touch OS:** Ubuntu Touch es un sistema operativo móvil basado en Linux y desarrollado por la compañía Canonical. Fue presentado el 2 de enero de 2013 al público. Las aplicaciones para este sistema operativo pueden ser programadas en varios lenguajes, tales como C++, QML, o HTML5 y el software se distribuye bajo una licencia GNU GPL la cual garantiza a los usuarios finales (personas, organizaciones, compañías) la libertad de usar, estudiar, compartir (copiar) y modificar el software. En la actualidad ningún fabricante vende dispositivos con Ubuntu Touch instalado, aunque compañías como la española Bq y la china Meizu planean vender terminales con Ubuntu Touch a lo largo del 2014.

De entre todos los sistemas operativos mencionados, son cuatro los que sobresalen por encima del resto en términos de cuota de mercado: Android, iOS y Windows

Phone (figura 16), siendo Android el elegido para desarrollar la aplicación. En la siguiente tabla se presenta una comparativa de cuatro de los nueve sistemas mencionados, dejando de lado los cinco restantes debido a que el soporte a la tecnología BLE en estos sistemas es inexistente o aún se encuentra en fase de pruebas. Dado la gran cantidad de datos disponibles, se han sintetizado las principales características utilizando una tabla para representar la información. De esta forma resulta más sencillo comparar las plataformas.

	Apple iOS 7	Android 4.3	Windows Phone 8	BlackBerry OS 7
Compañía	Apple	Open Handset Alliance	Microsoft	RIM
Núcleo del SO	Mac OS X	Linux	Windows NT	Mobile OS
Licencia de *software*	Propietaria	Software libre y abierto	Propietaria	Propietaria
Año de lanzamiento	2007	2008	2010	2003
Fabricante único	Sí	No	No	Sí
Variedad de dispositivos	modelo único	muy alta	media	baja
Soporte memoria externa	No	Sí	Sí	Sí
Motor del navegador web	WebKit	WebKit	Pocket Internet Explorer	WebKit
Soporte Flash	No	Sí	No	Si
HTML5	Sí	Sí	Sí	Sí
Tienda de aplicaciones	App Store	Google Play	Windows Marketplace	BlackBerry App World
Número de aplicaciones	825.000	850.000	160.000	100.000
Coste publicar	$99 / año	$25 una vez	$99 / año	sin coste
Actualizaciones automáticas del S.O.	Sí	depende del fabricante	depende del fabricante	Sí
Familia CPU soportada	ARM	ARM, MIPS, Power, x86	ARM	ARM
Máquina virtual	No	Dalvik	.net	Java
Aplicaciones nativas	Siempre	Sí	Sí	No
Lenguaje de programación	Objective-C, C++	Java, C++	C#, muchos	Java
Plataforma de desarrollo	Mac	Windows, Mac, Linux	Windows	Windows, Mac

Tabla 5. Comparativa de las principales plataformas móviles. Fuente [29]

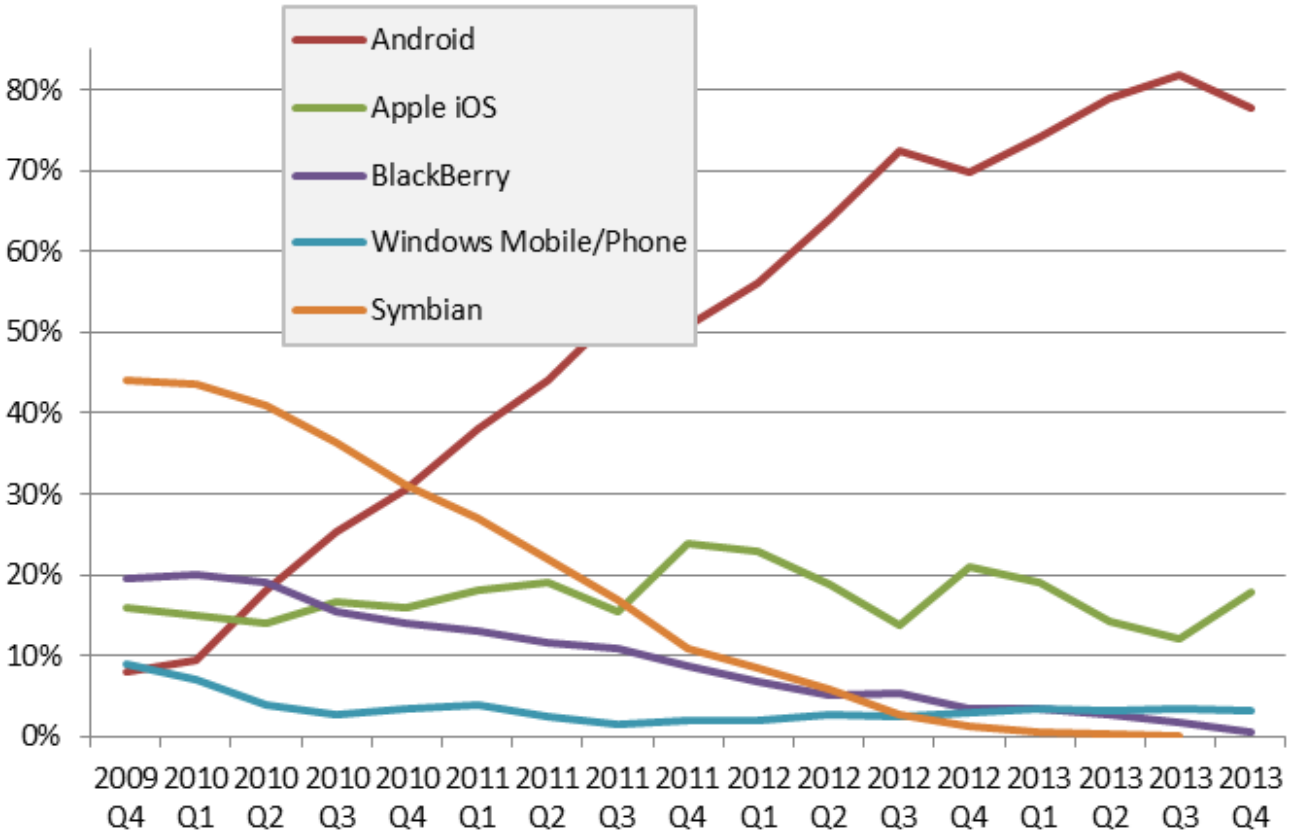


Figura 16. Porcentaje de teléfonos inteligentes vendidos según su sistema operativo hasta el último cuarto del 2013 en el mundo. Fuente: Gartner Group.

En la figura 16 se puede ver un estudio realizado por la empresa Gartner Group, donde se muestra la evolución del mercado de los sistemas operativos para móviles según el número de terminales vendidos hasta finales del año 2013.

Razones por las que se ha escogido el sistema operativo Android

Android presenta una serie de características que lo hacen diferente. Es el primero que combina en una misma solución las siguientes cualidades:

- Adaptable a cualquier tipo de hardware. Android no ha sido diseñado exclusivamente para su uso en teléfonos y tabletas. Hoy en día puede encontrarse en relojes, cámaras, electrodomésticos y gran variedad de sistemas empotrados que se basan en este sistema operativo. También en tecnología vestible o “wearables”, dispositivos que se ajustan perfectamente al perfil de dispositivo que utiliza BLE. Esta característica beneficia directamente al sistema diseñado ya que permite en un futuro modificar la parte hardware del mismo sin tener que hacer grandes cambios en el software. Así mismo,

contrasta radicalmente con la estrategia de Apple. En iOS hay que desarrollar una aplicación para iPhone y otra diferente para iPad.

- Portabilidad asegurada. Las aplicaciones finales son desarrolladas en Java lo que asegura que podrán ser ejecutadas en cualquier tipo de CPU, tanto presente como futuro. Esto se consigue gracias al concepto de máquina virtual.

- Gran cantidad de servicios incorporados. Esto permite al sistema diseñado escalar en funcionalidades en un futuro. Por ejemplo, localización basada tanto en GPS como en redes, bases de datos con SQL, reconocimiento y síntesis de voz, navegador, multimedia.

- Aceptable nivel de seguridad. Los programas se encuentran aislados unos de otros gracias al concepto de ejecución dentro de una caja que hereda de Linux. Además, cada aplicación dispone de una serie de permisos que limitan su rango de actuación (servicios de localización, acceso a Internet, etc.). Aunque en la aplicación no se maneja información sensible, es importante conocer que no está expuesta en exceso a ataques exteriores.

- Optimizado para baja potencia y poca memoria. Android utiliza la Máquina Virtual Dalvik. Se trata de una implementación de Google de la máquina virtual de Java optimizada para dispositivos móviles.

- Soporte del protocolo BLE desde la versión 4.3 Jelly Bean. Android 4.3 (nivel de API 18) incorpora un soporte nativo para Bluetooth Low Energy y proporciona APIs (Interfaz de programación de aplicaciones) que las aplicaciones pueden utilizar para descubrir dispositivos, búsqueda de servicios y leer/escribir características.

4 Implementación del sistema

4.1 Comunicación entre el sensor de temperatura y el microcontrolador

Las comunicaciones entre microcontrolador y sensor de temperatura son llevadas a cabo mediante el bus de comunicaciones en serie I²C [30] ya que es el soportado por el TC74 [31]. Antes de proceder a explicar de manera detallada el proceso de implementación del protocolo de comunicaciones entre microcontrolador y sensor conviene exponer una breve introducción de que es y para qué sirve el bus de comunicaciones en serie I²C.

4.1.1. I²C

El bus I²C (cuyo nombre viene del término en inglés *Inter-Integrated Circuit,* Inter-Circuitos Integrados) es un bus de comunicaciones en serie diseñado por la compañía Philips Semiconductor a principios de los años 80 con el objetivo de reducir los costes de fabricación en sus productos pertenecientes a la categoría de electrónica de consumo. Al principio sus aplicaciones se limitaban a controlar el volumen y el contraste en radios y televisiones. Más de treinta años después, se ha convertido en un estándar de facto dentro de la industria pudiendo encontrar esta tecnología en casi cualquier producto de la electrónica de consumo así como en una amplia variedad de sistemas y productos industriales, médicos y militares.

A día de hoy Philips (ahora NXP Semiconductors) sigue actualizando la especificación del bus I²C, datando la última revisión del 4 de Abril del 2014 [32].

Antes de la aparición de la tecnología I²C, las comunicaciones entre circuitos integrados o chips se realizaban utilizando multitud de pistas impresas en las PCB (del inglés Printed Circuit Board) o mediante cables, requiriendo que los circuitos integrados dispusiesen de 24, 28 o más pines. La mayoría de estos pines eran utilizados para la transmisión de direcciones, selección, control y transmisión de datos en paralelo, dando lugar por lo general a comunicaciones de 8 bits en un solo ciclo de reloj. Gracias a la tecnología I²C, la misma operación puede realizarse utilizando solo 2 cables comunicando en serie (figura 17). Aunque la velocidad de transmisión de datos es más lenta, transmitiendo 1 bit por ciclo de reloj, esta es suficiente en la mayoría de casos en los que se precisa de este tipo de comunicaciones.

Los dos cables o líneas que utiliza el bus I²C para transmitir la información son: SCL, el cual transmite señal de reloj sincronizando emisor y receptor durante la transferencia, y SDA, el cual transmite la señal de datos. También es necesaria una tercera línea, pero esta sólo es la referencia (GND o tierra). Como suelen comunicarse circuitos en una misma placa que comparten una misma masa esta tercera línea no suele ser necesaria.

Gracias a que el bus I²C permite comunicar en serie utilizando solo 2 líneas para la transmisión de datos en comparación con la transmisión de datos en paralelo, en la cual el número de líneas es mayor, el coste y el tamaño de los circuitos integrados se ve reducido. Como consecuencia de esta reducción los circuitos impresos o PCB en los que suelen ir montados los chips o circuitos integrados también se ven reducidos en tamaño y coste. Esto se debe a que al necesitar cada chip un número menor de cables o, en este caso, pistas para comunicar con otros, el tamaño, complejidad y costes de cada circuito impreso se ve por tanto reducido.

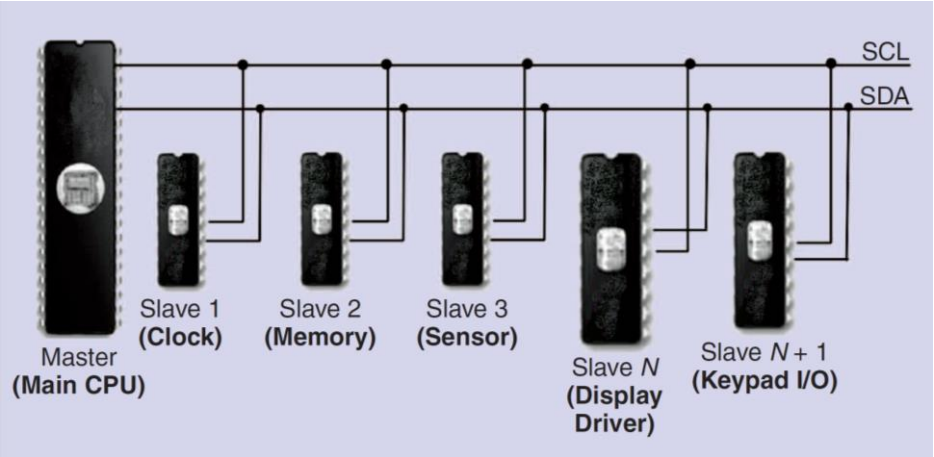


Figura 17. Sistema I²C básico compuesto por un dispositivo maestro (Main CPU) y múltiples chips esclavos. Fuente[33]

Como era de esperar no tardaron en surgir adaptaciones y versiones modificadas de la tecnología I²C partiendo siempre del bus de dos cables establecido por Philips, a continuación se exponen algunas de las más relevantes:

- **Access.bus:**

El Access.bus o A.b añade dos pines adicionales que se utilizan para alimentar a los dispositivos (+5 V y GND) pero limita el rango de direcciones a 125 dispositivos en lugar de los 1024 disponibles mediante I²C (ver sección **Múltiples dispositivos**). Además, solo soporta el modo estándar funcionando a 100 kbit/s o a baja velocidad a 10 kbit/s. Más adelante se explicaran los diferentes modos de funcionamiento del protocolo I²C. Similar al USB, debido a que permite a un dispositivo ser añadido a un ordenador sobre la marcha, nunca consiguió la popularidad de este último y, a pesar de ser lanzado con anterioridad, desapareció cuando la compañía Intel incluyo la tecnología USB en sus chips de control de la placa base estándar.

- **SMBus:**

El SMBus fue desarrollado por Intel en 1995. Basado en el protocolo I²C, funciona a una frecuencia de reloj de 10–100 kHz. A pesar de que los niveles de voltaje y los tiempos son marcados de manera más estricta que en el bus I²C, los dispositivos que soportan ambos protocolos utilizan el mismo bus para ambos como es el caso del TC74.

- **PMBus**

El PMBus es una variante del SMBus dirigido al control digital de fuentes de alimentación. Funciona a una frecuencia de reloj de 10–400 kHz y al igual que el SMBus, es un protocolo de comunicación de dos cables basado en el I²C. A diferencia de los otros dos protocolos, el PMBus tiene predefinidos una serie de comandos con el fin de garantizar que diferentes dispositivos funcionen de manera idéntica en lugar de especificar como comunicar los comandos definidos por el fabricante en función del dispositivo. Estos comandos abarcan características como los niveles de voltaje y corriente medidos, temperaturas, velocidad del ventilador, etc.

4.1.1.1 Características

A pesar del ahorro en costes que supone el uso de la tecnología I²C, el cual debería haber bastado para garantizar el éxito de la misma, el bus también presenta unas características que lo han convertido en el estándar que es, más de 30 años después de su nacimiento.

Jerarquía maestro – esclavo

Los dispositivos I²C son clasificados como maestro o esclavo. Un dispositivo que inicia la transmisión de un mensaje se define como maestro mientras que un dispositivo que solo responde al mensaje es esclavo. Dependiendo del tipo de dispositivo y de las especificaciones del fabricante, existen dispositivos que pueden ser solo maestros, solo esclavos, o pueden intercambiar los roles según lo requiera la aplicación.

Múltiples dispositivos

Cada dispositivo I²C esclavo dispone de una dirección única. Cuando un dispositivo maestro envía un mensaje, incluye la dirección del dispositivo al que se lo envía en el comienzo del mensaje. Todos los dispositivos esclavos conectados al bus “oirán” el mensaje, pero solo aquel que tenga la dirección especificada responderá al mensaje.

Lo más común en los dispositivos en el bus I²C es que utilicen direcciones de 7 bits, existiendo también dispositivos que utilizan direcciones de 10 bits. Una dirección de 7 bits implica que se pueden poner hasta 128 dispositivos sobre un bus I²C. Cuando se

envían las direcciones de 7 bit se utilizan transmisiones de 8 bits. El bit extra se utiliza para informar al dispositivo esclavo si el dispositivo maestro va a escribir o va a leer datos en él. La dirección de 7 bit se coloca en los 7 bits más significativos del byte y el bit de lectura/escritura en el bit menos significativo.

En cuanto al direccionamiento de 10 bits, este expande el número de posibles direcciones, permitiendo hasta 1024 dispositivos conectados al mismo bus. Los dispositivos con direcciones de 7 bits y los de 10 pueden coexistir dentro del mismo bus y ser utilizados a cualquier velocidad de bus. El direccionamiento de 10 bits se forma con los dos primeros bytes que siguen a una condición de inicio o *start.* Los primeros 7 bits del primer byte son la combinación de 111 0XX donde XX son los 2 bits más significativos de la dirección de 10 bits. El octavo bit del primer byte es el bit de lectura/escritura al igual que el direccionamiento de 7 bits. En el segundo byte se encuentran los 8 bits restantes de la dirección de 10 bits.

A pesar de que existen 9 posibles combinaciones de la dirección 1111 XXX, solo las cuatro combinaciones 1111 0XX se utilizan en la actualidad para el direccionamiento de 10 bits. Las cuatro restantes 1111 1XX se reservan para futuras revisiones del bus.

En la figura 18 se observa el aspecto de una transmisión de una dirección de 10 bits a un dispositivo esclavo. Después del primer byte cada dispositivo esclavo compara su propia dirección con los siete primeros bits, siendo posible que alguno de ellos genere un ACK (A1) debido a la coincidencia de los mismos. Todos los esclavos que encontraron una coincidencia comparan entonces los 8 bits del siguiente byte, dándose entonces solo una posible coincidencia que será el dispositivo al que iba dirigida la transmisión, el cual generara un segundo ACK (A2) que establecerá que la comunicación es firme y estable entre maestro y esclavo.

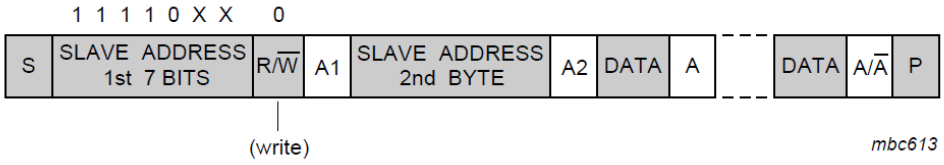


Figura 18. Un dispositivo maestro transmite una dirección de 10 bits a uno esclavo. Fuente[32]

Soporte para varios dispositivos maestros

El bus I²C permite que coexistan varios dispositivos maestros en el mismo bus. Para poder soportar varios maestros a la vez, el bus I²C debe ser capaz de controlar los posibles conflictos que se produzcan en las transmisiones si dos o más dispositivos maestros quieren "hablar" al mismo tiempo. Esta funcionalidad, llamada *bus arbitration loss detection*, permite a un dispositivo maestro detectar cuando se está produciendo un conflicto en las transmisiones y por tanto, terminando su uso del bus despejando el camino al mensaje generado por otro dispositivo maestro para que este cruce el bus sin alteraciones.

A partir de la condición inicial de bus libre, la cual se da cuando las señales SCL y SDA están en estado lógico alto, cualquier dispositivo maestro puede ocupar el bus, estableciendo la condición de inicio o *start* (figura 19). Esta condición se presenta cuando un dispositivo maestro pone en estado bajo la línea de datos (SDA), pero dejando en alto la línea de reloj (SCL).

Suponiendo que en el bus coexisten dos dispositivos maestros, y uno de ellos no ha recibido correctamente la condición de inicio establecida por el otro maestro, por lo que todavía piensa que el bus se encuentra libre. Este dispositivo podrá comprobar fácilmente si el bus está ocupado intentando poner ambas señales en estado lógico alto.

Debido a la arquitectura del bus (ver sección **Capa física**), cuando un dispositivo mantiene una línea en estado lógico bajo esta se mantiene bajo, por lo que el segundo dispositivo maestro no podrá poner nunca a ´1´ la línea de datos, estableciendo que el bus se encuentra ocupado en ese momento y esperando hasta que una condición de parada o *stop* (figura 19) se de en el bus, lo cual notifica que la anterior transmisión ha finalizado.

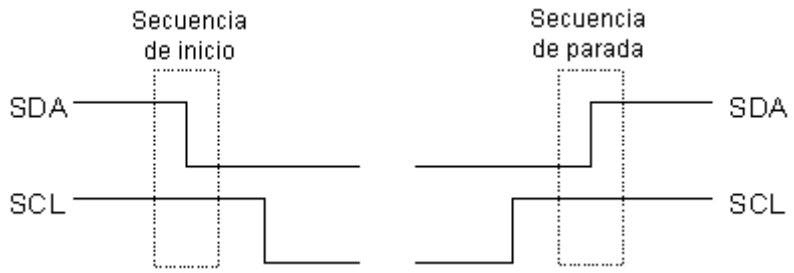


Figura 19. Secuencias de inicio y de parada.

Bits ACK y NACK

Después de la condición de inicio o *start* anteriormente citada, el dispositivo que controla el bus transmitirá un byte que contiene siete bits los cuales componen la dirección del dispositivo que se desea seleccionar, y un octavo bit que corresponde a la operación que se quiere realizar con él (lectura o escritura).

Si el dispositivo cuya dirección corresponde a la que se indica en los siete bits (A0-A6) está presente en el bus, contestará poniendo la línea de datos SDA en un estado lógico bajo, ubicado inmediatamente después del octavo bit (durante el noveno ciclo de reloj) que ha enviado el dispositivo maestro. Esta operación se define como bit de reconocimiento (ACK) y le indica al dispositivo maestro que el esclavo reconoce la solicitud y está en condiciones de comunicarse. Aquí la comunicación se establece en firme y comienza el intercambio de información entre los dispositivos. Existe la posibilidad de que en vez de un ACK (*acknowledge)* se genere un NACK (*Not Acknowledge)*, el cual consiste en mantener la línea de datos SDA en nivel alto durante el noveno pulso de reloj. Existen cinco situaciones que pueden provocar la generación de un NACK en una transferencia:

1) No existe ningún dispositivo en el bus con la dirección indicada por lo que nadie puede responder al mensaje con un ACK.

2) El dispositivo receptor no puede recibir o transmitir en ese momento no estando preparado aún para comunicar con el dispositivo maestro.

3) Durante la transferencia, el receptor recibe datos o comandos que no puede entender.

4) Durante la transferencia, el receptor no puede recibir más bytes de datos.

5) Un dispositivo maestro no ha finalizado el final de una transferencia con el dispositivo receptor.

Capa física

Uno de las características más importantes del diseño del bus I²C es su arquitectura basada en circuitos de salida de tipo drenador (o colector) abierto, lo que unido a que cada línea del bus suele llevar una resistencia de *pull-up* (R en la figura 20), lo cual provoca que el estado lógico normal de las líneas sea alto, permite realizar la función lógica AND entre las salidas conectadas al bus. Por lo tanto, para que una línea del bus sea puesta a 1, todos los transistores de salida deben estar cortados; en cambio, cuando un transistor de salida se satura, pone un 0 en la línea correspondiente y con ello ese dispositivo "domina" la línea del bus.

Además, los terminales de los dispositivos conectados al bus son, a la vez, entradas y salidas, de modo que cada línea del bus es bidireccional.

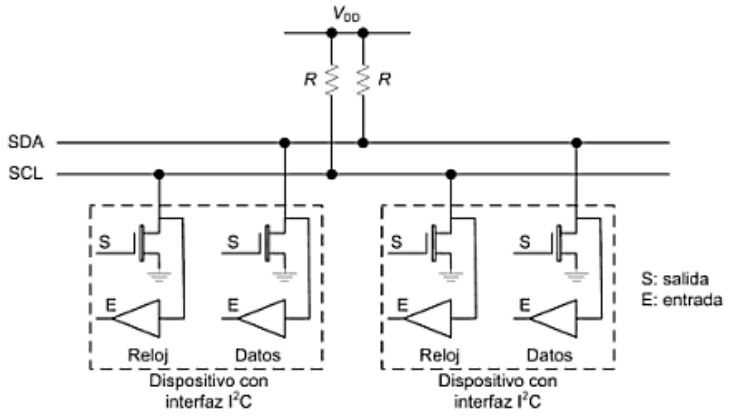


Figura 20. Arquitectura del bus I²C drenador/colector abierto.

Un aspecto adicional de esta funcionalidad es que un dispositivo esclavo es capaz de mantener la línea de reloj SCL en un nivel bajo, incluso después de que el maestro la haya cambiado a nivel alto. Esto introduce otro concepto asociado al bus I²C, el *clock stretching* (ver sección **Clock Stretching**). Mediante este mecanismo los dispositivos

esclavos pueden acceder a tiempo adicional pudiendo procesar o acceder a más información antes de que sea transmitida al maestro.

Protocolo de comunicaciones

El bus I²C presenta un protocolo bien definido a la hora de transmitir información a través del bus. En primer lugar toda transmisión comienza con una condición de inicio o *start* por parte de un dispositivo maestro y termina con una condición de parada o *stop*. Cada operación de lectura o escritura comienza con una condición de inicio o *start* seguida del byte que contiene la dirección del esclavo y el bit que define el tipo de operación a realizar, enviando un bit por cada ciclo de reloj.

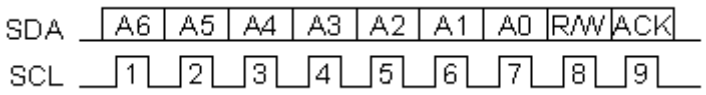


Figura 21. Envío de bit por ciclo de reloj.

Después del primer *start* es posible que se den otros *start* en la transmisión sin estar precedidos por un *stop*. De esta manera los esclavos conocen que la siguiente transferencia es parte del mismo mensaje. En la figura 22 se puede apreciar que aspecto tiene una transferencia I²C completa. Después de la condición de inicio o *start* se transmiten los 6 bits que contienen la dirección y el séptimo bit que contiene el tipo de operación a realizar.

Después de ello el dispositivo esclavo genera un bit de ACK lo cual da pie al dispositivo maestro para enviar 1 byte de datos el cual a su vez es respondido por otro ACK por parte del esclavo y repite la operación. Finalmente, el dispositivo maestro genera una condición de parada (SDA de nivel bajo a nivel alto mientras SCL se encuentra en nivel alto) indicando que la transmisión ha concluido.

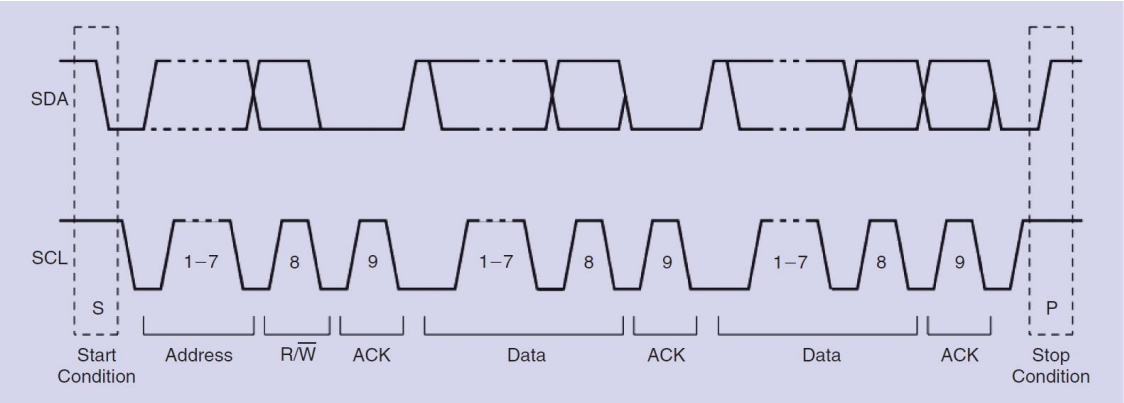


Figura 22. Una transferencia multibyte I²C completa. Fuente [32]

Clock stretching

Como se comentó en apartados anteriores, un dispositivo esclavo es capaz de mantener la línea de reloj SCL en un nivel bajo después de recibir (o enviar) un bit, indicando que aún no se encuentra listo para procesar más datos. El dispositivo maestro que está comunicando con él no finalizará la transmisión del bit actual hasta que la línea de reloj SCL vuelva a un nivel alto. Una vez que el maestro observe que la línea de reloj SCL está en nivel alto de nuevo, se dará por concluida la transmisión de ese bit en particular.

Aunque un dispositivo maestro es capaz como es natural de mantener la línea de reloj en un nivel bajo, el término *clock stretching* se reserva normalmente al uso que hacen los esclavos. Además, aunque en teoría cualquier ciclo de reloj puede ser utilizado para hacer *clock stretching,* suelen ser utilizados los intervalos antes o después del bit de ACK para tal uso. Por ejemplo, si el esclavo es un microcontrolador, este mantendrá la línea de reloj baja después de cada byte recibido para que el software disponga del tiempo necesario para validar el byte entrante y enviar un ACK o, por el contrario, un NACK. Solo mediante *Clock streching* un esclavo puede manejar la línea de reloj SCL. Merece la pena destacar que algunos dispositivos no soportan el *clock streching,* por lo general estos dispositivos se etiquetan como TWI (two-wire interface) y no I²C.

Modos de operación en el bus

El bus puede operar de 4 maneras distintas según el tipo de dispositivo que realice las comunicaciones y el sentido de las mismas:

1) El maestro inicia la transmisión: Un nodo maestro envía datos a uno esclavo.

2) El maestro recibe: Un nodo maestro recibe datos desde uno esclavo.

3) El esclavo transmite: Un nodo esclavo envía datos a uno maestro.

4) El esclavo recibe: Un nodo esclavo recibe datos desde uno maestro.

En todos los casos el nodo maestro maneja la señal de reloj SCL, y el esclavo actúa en consecuencia. Si el nodo maestro está transmitiendo, este genera el flujo de bits a transmitir por SDA hacia el esclavo. Los diferentes modos de funcionamiento unidos permiten a un circuito integrado participar en sistemas más complejos con varios maestros y en rol de maestro o esclavo. Estas funcionalidades son fundamentales si existen múltiples redes de dispositivos conectados mediante I²C dentro del mismo sistema. Así mismo, también se utilizan con los denominados *I²C bus extenders*, los cuales permiten que el bus opere a lo largo de decenas o incluso cientos de metros. En electrónica de consumo el coste suele ser el factor determinante, por lo que la mayoría de circuitos integrados utilizados actúan a modo de maestro o de esclavo, estando generalmente limitados a una sola función, por ejemplo transmitir datos (p.ej un sensor) o recibiendo comandos (p.ej. un microcontrolador).

Velocidades de bus

En su origen, el bus I²C estaba limitado a operaciones con velocidades no superiores a los 100 kbit/s. Sin embargo a lo largo del tiempo se ha ido aumentando esa cifra añadiendo diferentes modos hasta llegar a los 6 que existen en la actualidad: Standard, Fast-mode (Fm), Fast-mode Plus (Fm+), High-speed mode (Hs-mode) y Ultra Fast-mode.

Todos los dispositivos I²C presentan compatibilidad reversible entre los diferentes modos, por lo que cualquier dispositivo podrá funcionar a velocidades de bus inferiores. Existe una excepción a esto último, ya que en el caso de dispositivos que

soportan el Ultra Fast-mode no exista esta compatibilidad reversible debido a que solo se da en un bus unidireccional, al contrario que el resto de modos limitados a buses bidireccionales. A continuación se muestran los diferentes modos que existen así como el tipo de bus que requieren para su funcionamiento.

• Bus bidireccional:

– Standard-mode (Sm), con un bit rate de hasta 100 kbit/s.

– Fast-mode (Fm), con un bit rate de hasta 400 kbit/s.

– Fast-mode Plus (Fm+), con un bit rate de hasta 1 Mbit/s.

– High-speed mode (Hs-mode), con un bit rate de hasta 3.4 Mbit/s.

• Bus unidireccional:

– Ultra Fast-mode (UFm), con un bit rate de hasta 5 Mbit/s.

4.1.2. Implementación del protocolo de comunicaciones entre microcontrolador y sensor de temperatura

A la hora de implementar las comunicaciones vía I²C entre el microcontrolador y el sensor de temperatura, en primer lugar se debe localizar donde se efectuarán las conexiones tanto en la placa de desarrollo como en el sensor. En el caso de la placa estas dependerán de la solución software mediante la cual se quieran realizar las comunicaciones, ya sea una solución que utilice la librería nativa para I²C de Arduino (1) o una solución software mediante *bit banging* (2) [34].

4.1.2.1 Opción 1. Utilizando la librería Wire para I²C de Arduino

Conexionado

Si se opta por la opción (1) y se utiliza la librería nativa para I²C de Arduino llamada Wire, las conexiones de la línea de datos SDA y de la línea de reloj SCL deberán realizarse en los pines SDA y SCL respectivamente, los cuales se pueden ver en la figura 23.

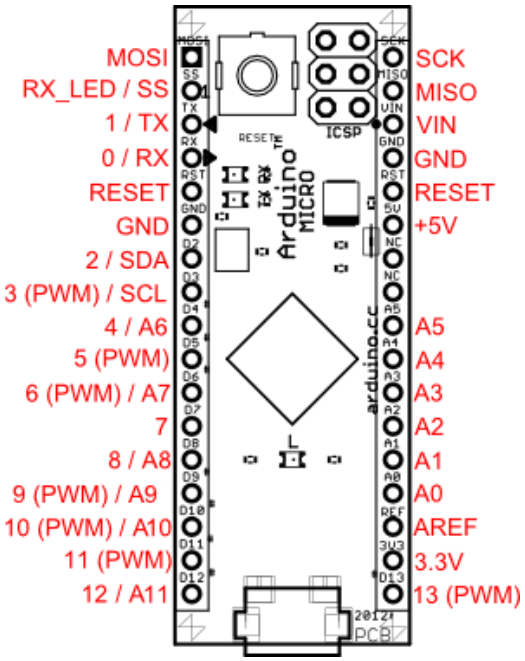


Figura 23. Mapeado de pins de la placa de desarrollo Arduino Micro. Fuente [35]

Por parte del TC74, una vez conectadas las líneas SDA y SCL con sus respectivos pines en la placa de desarrollo, se conectará la patilla de tierra GND a cualquiera de los pines GND de la placa y la patilla V_{DD} al pin de 5V.

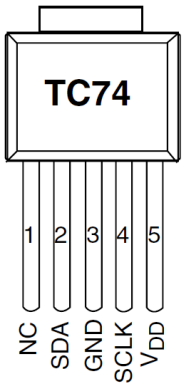


Figura 24. Patillaje del TC74. Fuente [31]

Soporte hardware y software

En cuanto al soporte software en este caso, la librería Wire de Arduino permite interactuar con el hardware dedicado I²C del microcontrolador, el cual permite por ejemplo despertar al micro si se encuentra en modo *sleep* al recibir una dirección valida o proteger las líneas del bus ante picos de voltaje [36]. El microcontrolador incorporado en la placa Arduino, el ATmega32u4, dispone de un módulo destinado a

las comunicaciones I²C llamado módulo TWI (*Two Wire Interface)*. La figura 25 muestra un esquema de las partes de las que se compone el modulo y la relación entre ellas.

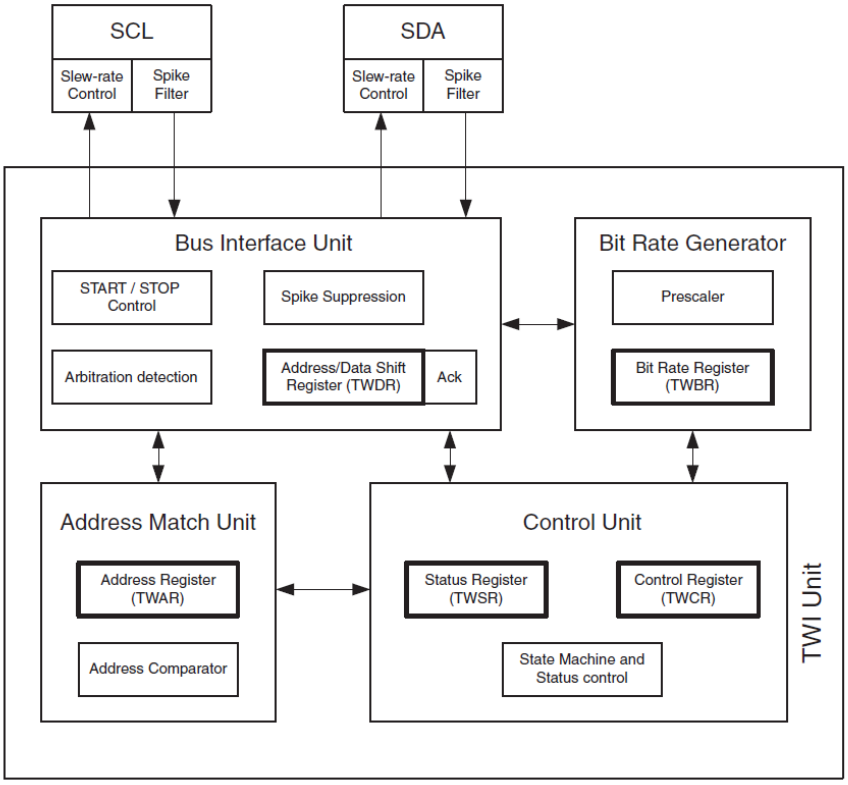


Figura 25. Diagrama de bloques del módulo TWI. Página 229 de la datasheet. Fuente [36]

Como se puede observar en la figura 25 el módulo TWI incorporado en el ATmega32u4 dispone de cinco registros, los cuales son utilizados en las comunicaciones como se explica a continuación.

Registro	**Nombre**	**Función**
TWCR	Control Register	Controla las acciones del módulo TWI.
TWSR	Status Register	Muestra el estado de las acciones realizadas por el módulo TWI.
TWDR	Address/Data Shift Register	Contiene los datos a transmitir o los que se han recibido.
TWBR	Bit Rate Register	Controla la frecuencia de la línea de reloj (SCL)
TWAR	Adress Register	Contiene la dirección del dispositivo

Tabla 6. Registros del módulo TWI. Fuente [36]

El microcontrolador está diseñado para manejar las comunicaciones I²C de la siguiente forma: Primero se configuran y preparan los registros y después el módulo TWI realiza las comunicaciones. Una vez finalizada la transmisión el módulo TWI envía una interrupción al procesador para notificar cualquier cambio (operación realizada con éxito, error, etc.). En la figura 26 se puede observar un diagrama de flujo en el cual se ejecuta una operación de escritura en el bus I²C.

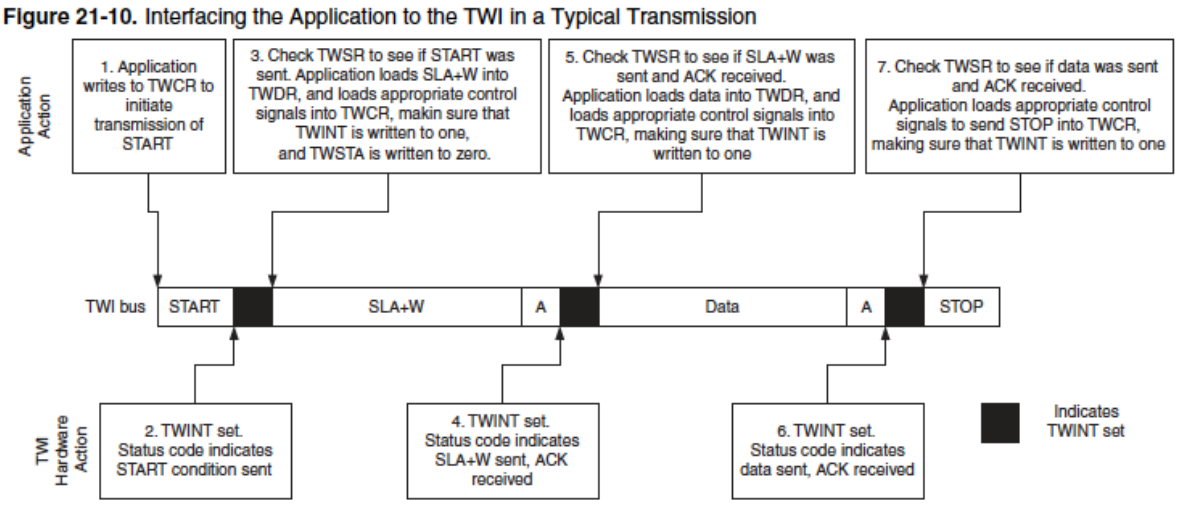


Figura 26. Operación de escritura en el bus I²C utilizando el módulo TWI. Fuente [36]

Este diagrama de flujo sirve como ayuda visual para entender cómo trabaja el microcontrolador a la hora de efectuar una transmisión I²C y como la librería Wire interacciona con el mismo para ejecutarla con éxito.

En primer lugar utilizando la librería la aplicación software configura el registro TWCR para generar una condición de inicio o *start.* El micro prueba que el bus se encuentre libre (que SDA y SCL se encuentren a nivel alto) y escribe un valor específico en el registro TWCR, ordenando al hardware TWI que transmita un *start.* Es importante que el bit TWINT se encuentre con un valor lógico 1, o de lo contrario el hardware TWI no iniciará ninguna transmisión. Esta operación con el bit TWINT se efectuará de manera recurrente a lo largo de la transmisión ya que marcara cada interrupción producida después de cada operación, como se puede apreciar en la figura 26 después de cada bloque de color negro.

Después de que la condición de inicio haya sido transmitida, el hardware provocará una interrupción como se ha descrito y el registro TWSR se actualizará con un código de estado que indica que la condición de inicio ha sido transmitida con éxito.

En el siguiente paso la aplicación software comprueba el valor del registro TWSR, asegurando que el valor sea el correcto. Si el código de estado es correcto la aplicación cargará el registro TWDR con la dirección del dispositivo esclavo y con el bit de escritura / lectura (en este caso escritura). Este valor es conocido como SLA+W (del inglés *Slave Address plus Write*). A continuación se carga en el registro TWCR el valor que indica que se debe iniciar la transmisión de SLA+W. Utilizando la librería la aplicación software comprobará que la transmisión se ha efectuado de forma correcta asegurando que el código de estado en el registro TWSR se ha actualizado indicando que el paquete SLA+W se ha transmitido con éxito. Este código también mostrara si el dispositivo esclavo ha respondido con un ACK o no.

Si el código de estado es correcto la aplicación software utilizando la librería cargará el registro TWDR con la información a transmitir, actualizando el valor del registro TWCR para indicar al hardware TWI que debe iniciar la transmisión de nuevo.

Cuando el paquete haya sido transmitido, el registro TWSR se actualizará con el código de estado que indicará si el paquete se ha transmitido con éxito. Este código también mostrara si el dispositivo esclavo ha respondido con un ACK o si por el contrario ha respondido con un NACK.

Por último la aplicación software comprobará el valor del registro TWSR y si todo es correcto actualizará el registro TWCR indicando que se debe transmitir una condición de parada o *stop,* finalizando la transmisión.

Aplicación software

Como se ha podido observar en el apartado anterior, utilizando la librería Wire de Arduino en la aplicación software que controla las comunicaciones con el sensor se interactúa directamente con el hardware dedicado del microcontrolador para las comunicaciones I²C. En este apartado se mostrará mediante pseudocódigo la aplicación programada mediante el entorno de desarrollo integrado (IDE) de Arduino y la librería Wire, mostrando las funciones utilizadas de la librería así como explicando en detalle su uso y características.

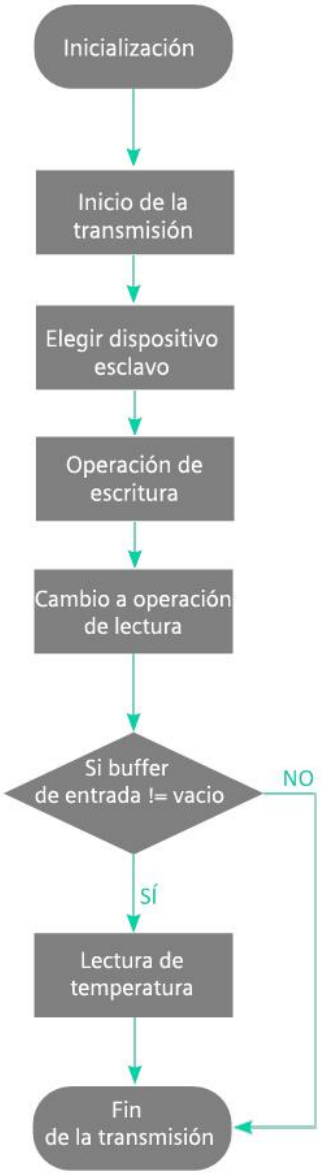


Figura 27. Diagrama de flujo de la aplicación software basada en la Librería Wire.

Inicialización

En esta etapa del programa se inicializan todas las variables que se utilizarán a lo largo del mismo, se incluirán las librerías necesarias, etc. Entre las variables inicializadas se encuentra:

- Dirección del dispositivo esclavo (sensor TC74): Variable que almacena la dirección del sensor en formato hexadecimal. Según el encapsulado del sensor (figura 28) se obtiene la versión del sensor utilizado, que en este caso corresponde a la TC74A0-5.0VAT. Por tanto, buscando el módulo según la versión en la hoja de características del TC74 [31] se obtiene la dirección del dispositivo.

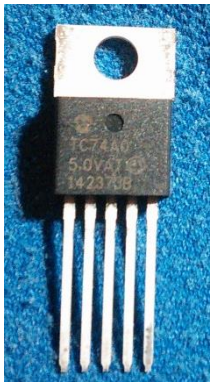


Figura 28. Versión del sensor.

Part Number	Package	Address
TC74A0-5.0VAT	TO-220-5	1001 000

Figura 29. Dirección del TC74. Fuente [31]

Inicio de la transmisión

A la hora de comenzar cualquier comunicación I²C la función begin() incluida en la librería Wire debe ser llamada. Mediante este método se llama a la función twi.init() del microcontrolador la cual:

- Activa el hardware del microcontrolador habilitando el acceso a los pines destinados al uso en las comunicaciones I²C.

- Inicializa la frecuencia de reloj a utilizar por el hardware a 100kHz, lo cual es el máximo permitido por el TC74 ([31] Página 4) y equivale al modo estándar visto en el apartado 4.1.1. en la sección **Velocidades de bus**.

Elegir dispositivo esclavo

Mediante la función beginTransmission(address) se prepara la transmisión de datos a la dirección del dispositivo indicada en el argumento de la función.

Operación de escritura

Gracias a la función write() es posible escribir el byte indicado en el argumento de la función en el dispositivo esclavo indicado por el maestro. En este caso escribe 0x00 lo cual indica al TC74 que se quiere leer en el registro de temperatura ([31] Página 8).

Cambiar a operación de lectura

La función requestFrom(adress,n) es utilizada cuando el microcontrolador actúa como maestro. Mediante esta función se llama al método twi_readFrom() del ATmega32u4 el cual permite ejecutar una operación de lectura enviando una condición de inicio seguida de una lectura de n bytes desde la dirección indicada como argumento, donde n es el número de bytes indicado en el segundo operador del argumento. Si se solicita una lectura de más bytes de los permitidos o disponibles, el esclavo devolverá un NACK y la función finalizará limpiamente.

Si se cumple condición, operación de lectura

Para establecer la condición se utiliza la función available(), la cual permite obtener el número de bytes disponibles en el buffer de entrada de la librería Wire accesibles con una llamada a la función read(). En este caso se cumplirá la condición simplemente cuando el buffer no se encuentre vacío, lo cual indicará que ha recibido algo por parte del sensor.

Lectura de temperatura

Mediante la funcion read() el dispositivo maestro lee un byte transmitido desde el dispositivo esclavo. En este caso guarda el byte leído como temperatura.

Fin de la transmisión

Una vez obtenida la temperatura, se invocará a la función endTransmission() de la librería Wire. Esta función tiene como argumento por defecto un booleano de valor *true*, enviando una secuencia de parada al bus y dejándolo libre para futuras transmisiones. Es posible no liberar el bus enviando otro valor (*false*), lo cual será interpretado por el bus como una nueva secuencia de inicio o *restart*.

4.1.2.2 Opción 2. Bit banging

Conexionado

Si se opta por la opción (2) y se implementa el protocolo I²C mediante *bit banging* [34], las conexiones de la línea de datos SDA y de la línea de reloj SCL podrán realizarse con cualquier pin digital IO disponible en el microcontrolador. En este caso se utilizarán los pines digitales 8 y 10 de la placa de desarrollo, que corresponden a los pines PB4 y PB6 del microcontrolador (figura 30).

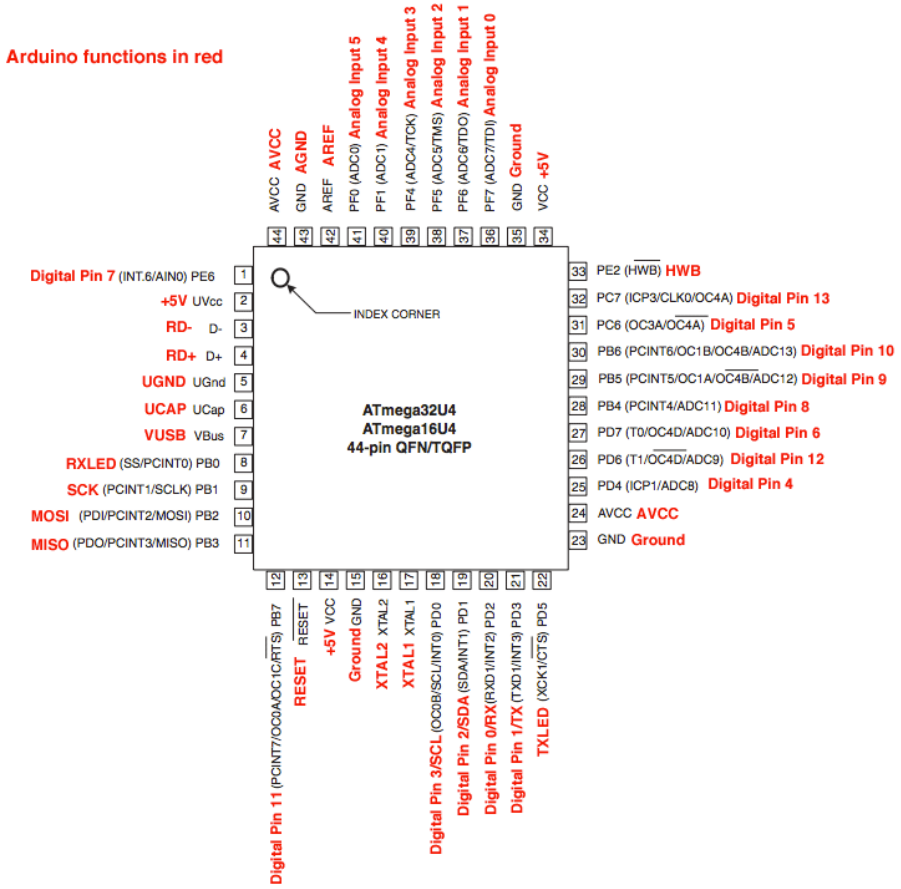


Figura 30. Mapeado de pines del microcontrolador ATmega32u4. Fuente [35]

Por parte del TC74, una vez conectadas las líneas SDA y SCL a sus respectivos pines en la placa de desarrollo, la conexión del pin GND y del pin V_{DD} se realizará de manera idéntica a lo expuesto en el conexionado del caso 1.

Soporte hardware y software

Debido a que ahora se va a implementar el protocolo únicamente mediante software, se deben hacer ciertas comprobaciones que antes no eran necesarias ya que se disponía de soporte hardware específicamente diseñado para comunicaciones I²C.

Como se comentó en la sección **Capa Física** del apartado 4.1.1, una de las características más importantes del diseño del bus I²C es su arquitectura basada en circuitos de salida de tipo drenador (o colector) abierto, lo cual provoca que el estado lógico normal de las líneas sea alto siempre y cuando se dispongan de resistencias de pull-up en ambas líneas (SDA y SCL). Como este era uno de los requisitos incluidos a la hora de elegir una placa de desarrollo para el sistema, el microcontrolador elegido dispone de estas resistencias accesibles para cualquier pin digital como se puede observar en la figura 31.

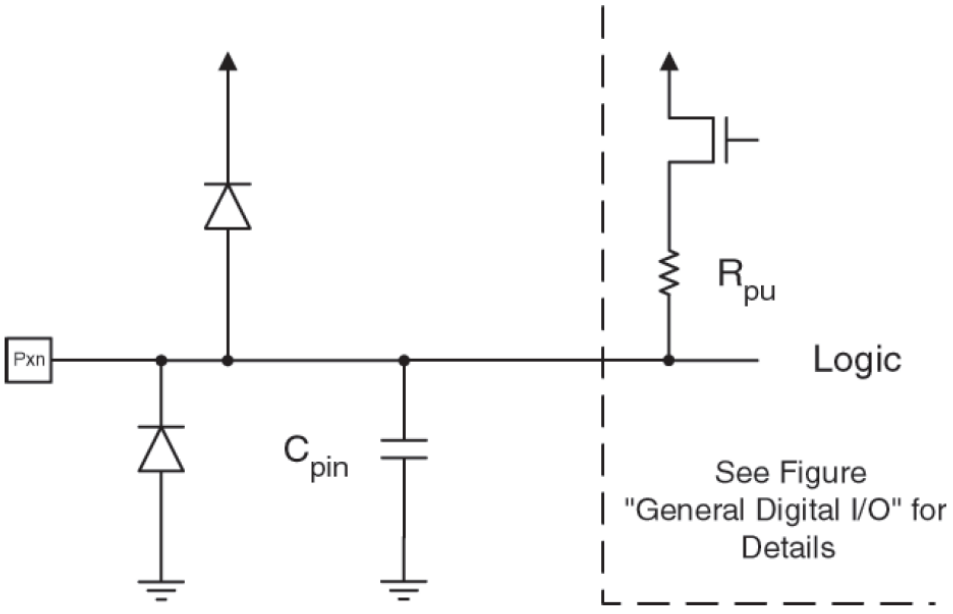


Figura 31. I/O Pin Equivalent Schematic. Fuente ([36] Página 65)

Los valores entre los cuales tienen que encontrarse estas resistencias para el correcto funcionamiento del bus en el modo estándar vienen determinados en la especificación

I²C de NXP ([32] Página 55), de donde se observa que el valor mínimo de la misma puede ser calculado según la ecuación:

$$Rp_{min} = \frac{V_{DD} - V_{OL(max)}}{I_{OL}}$$

Lo cual nos deja un valor tomando los datos suministrados por el fabricante ([36] Página 378) de 430 Ω. En cuanto al valor máximo de la resistencia pull-up, si se hace una estimación de la capacidad máxima del bus teniendo en cuenta que cada pin presenta como máximo 10 pF de capacidad [36], el número de pines utilizados en el bus así como la capacidad del resto de componentes incluyendo el sensor de temperatura, accediendo a la gráfica de la especificación I²C ([32] Página 55, figura 41) este presenta un valor máximo de 60 kΩ, por lo que el rango permitido por el microcontrolador se ajusta a ambos limites ([36] Página 378).

En segundo lugar hay que analizar si los valores de voltaje obtenidos en los pines digitales de la placa de desarrollo en nivel alto (valor 1 lógico) y nivel bajo (valor 0 lógico) son reconocidos por el sensor de temperatura como tales. Según la hoja de características del TC74 [31] el sensor reconocerá como 1 lógico cualquier voltaje entrante igual o superior a 0.8xV_{DD}, que en este caso equivale a 4V ya que el sensor estará alimentado a 5V como se ha explicado en la sección **Conexionado**. Por otro lado el sensor reconocerá como 0 lógico cualquier voltaje entrante igual o inferior a 0.2xV_{DD}, que en este caso equivale a 1V. Analizando la hoja de características se comprueba que en ambos casos se cumplen los requerimientos del TC74, obteniendo un voltaje en los pines de salida de 4.2V a nivel alto y 0.7V a nivel bajo.

Aplicación software

En este apartado se mostrará en primer lugar un diagrama de flujo de la aplicación programada mediante el entorno de desarrollo integrado (IDE) utilizando *bit banging* (figura 32). Implementar el protocolo I²C mediante *bit banging* implica programar cada una de las acciones a realizar por el microcontrolador sin ayuda hardware adicional, por lo que se debe controlar en todo momento el estado de las líneas SDA y SCL. Por ello se ha creado una librería propia en la que estarán definidas e implementadas todas las funciones utilizadas en la aplicación software.

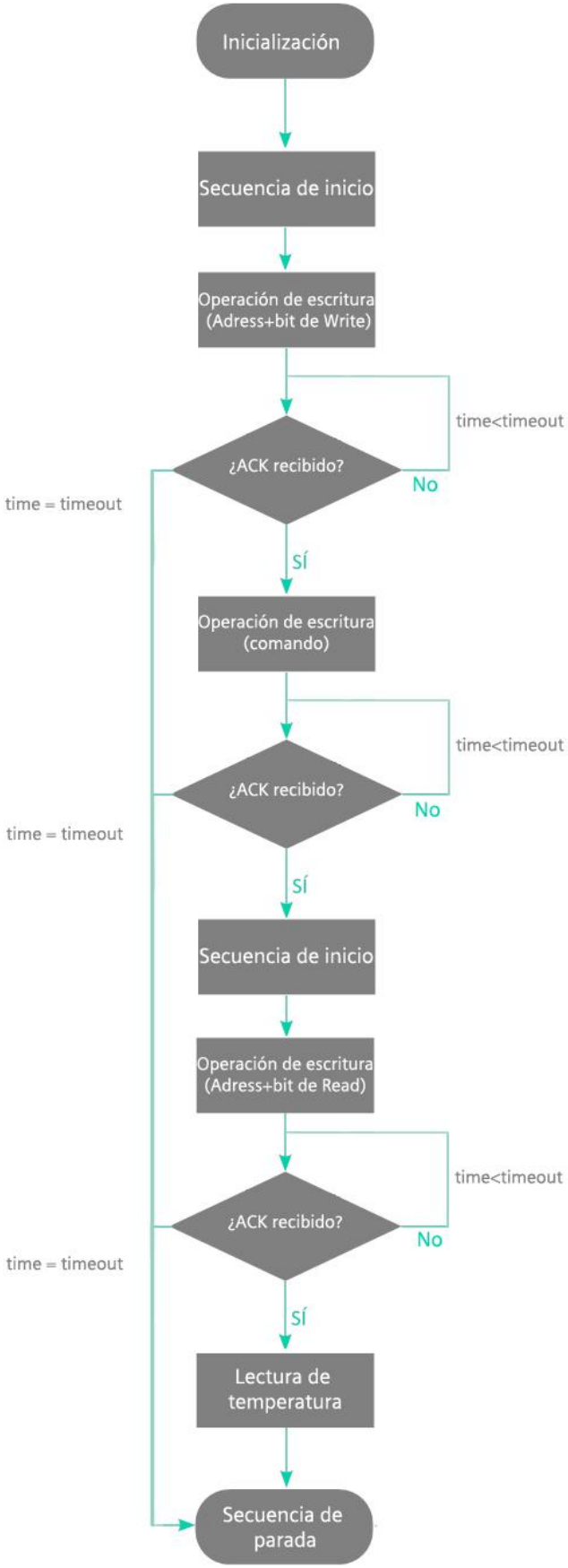


Figura 32. Diagrama de flujo de la aplicación software realizada mediante bit banging.

Inicio de la transmisión (Secuencia de inicio)

Para comenzar la transmisión se llama a la función i2c_start() la cual hace que el dispositivo maestro genere una condición de inicio comprobando antes que el bus se encuentre libre. Para generar la condición de inicio o *start* es necesario como se pudo ver en el apartado 4.1.1 que la línea de datos SDA pase de un nivel lógico 1 a un nivel lógico 0 mientras la línea de reloj SCL se mantiene a 1.

Operación de escritura (Adress+RW)

Después del *start* el siguiente paso es llamar a la función i2c_write(), la cual realizará una operación de escritura en el dispositivo esclavo del byte indicado en el argumento de la función. En este caso el byte a escribir es 1001 0000 (0x90h), es decir la dirección del sensor seguida de un bit que indica la operación a realizar, escritura en este caso (0). Esta función implementa además la lectura del bit ACK enviado desde el sensor, reiniciando la transmisión si se produce un *timeout* o se recibe un NACK.

Operación de escritura (0x00)

Posteriormente se vuelve a llamar a la funcion i2c_write() pero esta vez para enviar el byte que contiene el comando 0x00. Este comando provoca que el sensor TC74 llame a la función Read Temperature (TEMP) del mismo ([31] Página 9), la cual activará la medición continua de temperatura por parte del sensor guardando las medidas en el registro de temperatura interno del mismo (TEMPERATURE REGISTER (TEMP)) ([31] Página 9). Todas las medidas serán guardadas en formato binario de complemento a 2. Esta función implementa además la lectura del bit ACK enviado desde el sensor, reiniciando la transmisión si se produce un *timeout* o se recibe un NACK.

Restart

Después de la última operación de escritura se deberá volver a generar una condición de inicio o *start*, la cual es necesaria ya que “reinicia” la transmisión permitiendo al dispositivo maestro volver a indicar si desea realizar una operación de escritura o de lectura en el esclavo.

Preparando la lectura (Operación de escritura Adress+RW)

En este caso como se quiere leer la temperatura del registro se llamara a la función i2c_write() de nuevo, utilizando como argumento la dirección del sensor y el bit de escritura o lectura a 1 en formato hexadecimal (1001 0001 = 0x91h) indicando que se desea realizar una operación de lectura. Esta función implementa además la lectura del bit ACK enviado desde el sensor, reiniciando la transmisión si se produce un *timeout* o se recibe un NACK.

Operación de lectura

Una vez enviado el último byte se procederá a leer la temperatura mediante la función i2c_read(). Esta función leerá un bit en cada ciclo de reloj hasta completar 8 ciclos de reloj (1 byte) y guardará el valor en una variable accesible desde fuera de la función.

Final de la transmisión (Secuencia de parada)

Por último se llama a la función i2c_stop() la cual finalizará las comunicaciones con el sensor ejecutando una secuencia de parada, la cual como se pudo ver en el apartado 4.1.1 consiste en que la línea de datos SDA pase de un nivel lógico 0 a un nivel lógico 1 mientras la línea de reloj se mantiene a 1.

Razones por la que se ha escogido la opción 1

A pesar de que las dos opciones son válidas y cumplen su cometido a la perfección, a la hora de implementar el sistema es necesario elegir una de ellas. Por ello se analizarán las desventajas que presentan ambas opciones, empezando por primera.

Desventajas de la opción 1

Esta opción, la cual utiliza la librería Wire de Arduino, presenta una desventaja frente a la opción 2 y es que mientras el módulo TWI está realizando comunicaciones mediante el bus I^2C bloquea la aplicación, lo que significa que aunque saltase una interrupción o cualquier elemento que obligase a la aplicación a detener su actividad esta no lo haría hasta que el módulo TWI dejase de transmitir a través del bus.

Pese a ello, esta será la opción escogida, ya que el impacto de la desventaja expuesta en un sistema como el diseñado es mínimo, siendo el tiempo que el módulo pueda tardar en terminar las comunicaciones en el bus inferior en varios ordenes de magnitud al intervalo entre comunicaciones, además de no existir ninguna condición en la solución programada que pueda bloquear de manera indefinida la aplicación.

Desventajas opción 2

Mediante la técnica de *bit banging* no se utiliza ningún hardware dedicado del microcontrolador, lo cual puede ser una ventaja si no se dispone del mismo o se quiere tener un mayor control sobre el funcionamiento de la aplicación. Sin embargo, al tener que emular mediante software el protocolo completo se consumen más recursos del sistema y el microcontrolador gasta más tiempo leyendo o enviando señales a los pines, a costa de no poder dedicar ese tiempo a otras tareas.

4.2 Comunicación entre el microcontrolador y el módulo Bluetooth nRF2740

Las comunicaciones entre microcontrolador y módulo Bluetooth Low Energy nRF2740 son llevadas a cabo mediante el bus de comunicaciones en serie SPI ya que es el soportado por el chip nRF8001 [37]. Antes de proceder a explicar de manera detallada el proceso de implementación del protocolo de comunicaciones entre el microcontrolador y el módulo BLE conviene exponer una breve introducción de que es y para qué sirve el bus de comunicaciones en serie SPI.

4.2.1. SPI

SPI (del inglés Serial Peripheral Interface) es un bus de comunicaciones en serie desarrollado por la compañía Motorola, habiéndose convertido hoy en día en un estándar de facto dentro de la industria. Al contrario que para I^2C, el cual dispone de una especificación actualizada y accesible en todo momento, es difícil encontrar una especificación “oficial” del bus SPI, siendo lo más parecido a la misma una guía que la

división de Motorola de semiconductores (actualmente Freescale Semiconductors) realizó en el año 2000.

El bus SPI permite conectar de manera síncrona dos o más dispositivos en modo full-duplex, asumiendo estos los roles de maestro o esclavo. Siempre debe existir un dispositivo maestro el cual se encarga siempre de iniciar las comunicaciones y posee el control sobre las mismas. En una comunicación SPI no está limitado el número de dispositivos esclavos unidos al bus como en I²C, donde se restringe el mismo mediante el uso de direcciones limitadas, permitiendo conectar tantos dispositivos esclavos como las limitaciones hardware permitan. Además, al ser establecida una comunicación en modo full-duplex, el dispositivo maestro podrá enviar datos a los esclavos al mismo tiempo que los recibe. Para tal fin, el bus SPI tiene asociados 4 pines o líneas:

• Slave select ($\overline{SS}$)

• Serial clock (SCK)

• Master out/slave in (MOSI)

• Master in/slave out (MISO)

Transmisiones mediante el bus SPI

Cada transmisión dentro del bus SPI se basa en la utilización de registros de desplazamiento. Cada dispositivo, sea esclavo o maestro dispone de un registro de desplazamiento de 8 bits. El tamaño de estos registros de desplazamiento puede ser superior a 8 bits (10, 12 bits, etc.) pero siempre debe ser del mismo tamaño en el dispositivo esclavo y en el maestro. Ambos registros disponibles en maestro y esclavo están vinculados mediante las líneas MOSI y MISO formando un registro total de 16 bits.

Cuando se ejecuta una transmisión de datos, el registro de desplazamiento de 16 bits se desplaza 8 posiciones desde el dispositivo maestro hacia el esclavo, dando lugar a la transmisión de datos entre ambos. Dicho de otra forma, ambos registros forman un

circuito circular entre ambos dispositivos (figura 33), operando en un régimen SISO (Serial-In/Serial-Out). Así, la salida del registro de desplazamiento del dispositivo maestro está conectada a la entrada del registro de desplazamiento del esclavo, y la salida del registro de desplazamiento del dispositivo esclavo a la entrada del registro de desplazamiento del maestro.

Como se mencionó con anterioridad, el bus SPI permite conectar de manera síncrona dos o más dispositivos, lo que significa que debe existir una señal de reloj para sincronizar las transferencias.

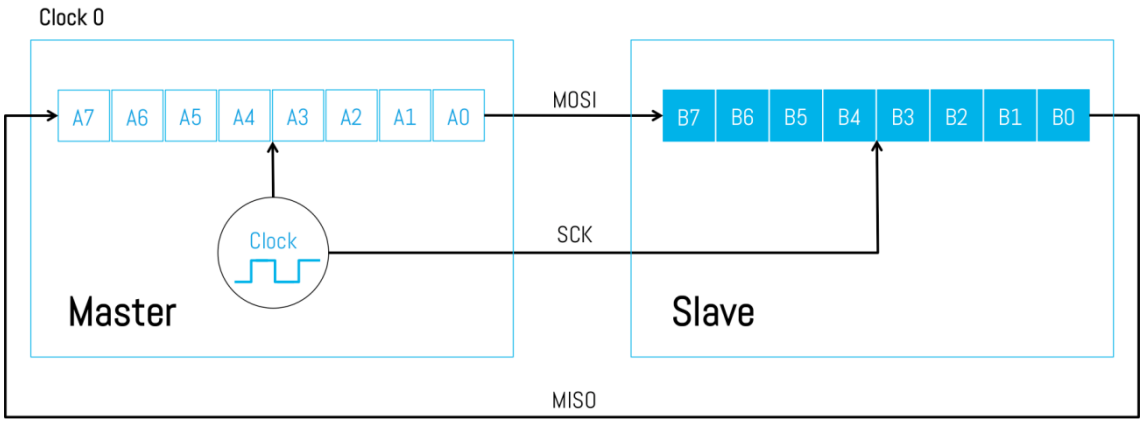


Figura 33. Registros de desplazamiento en el bus SPI. Fuente [38]

Transferencia de datos

Como se puede observar en la figura 33, se han numerado los bits dentro de cada registro, siendo A7 y B7 los bits más significativos (MSB) y A0-B0 los menos (LSB). Suponiendo que este sea el estado inicial en el que se encuentran ambos registros, en cuanto se inicia la transmisión y llega el primer pulso de reloj los registros comienzan a operar y comienza la transmisión del primer bit entre maestro y esclavo. En este caso el bit A0 es el primero en salir hacia el registro de desplazamiento del dispositivo esclavo, desplazando todos los bits del esclavo hacia la derecha y ocupando el primer lugar en el mismo. A su vez, el bit B0 desplaza todos los bits del registro de desplazamiento del dispositivo maestro hacia la derecha ocupando el primer lugar como se puede observar en la figura 34.

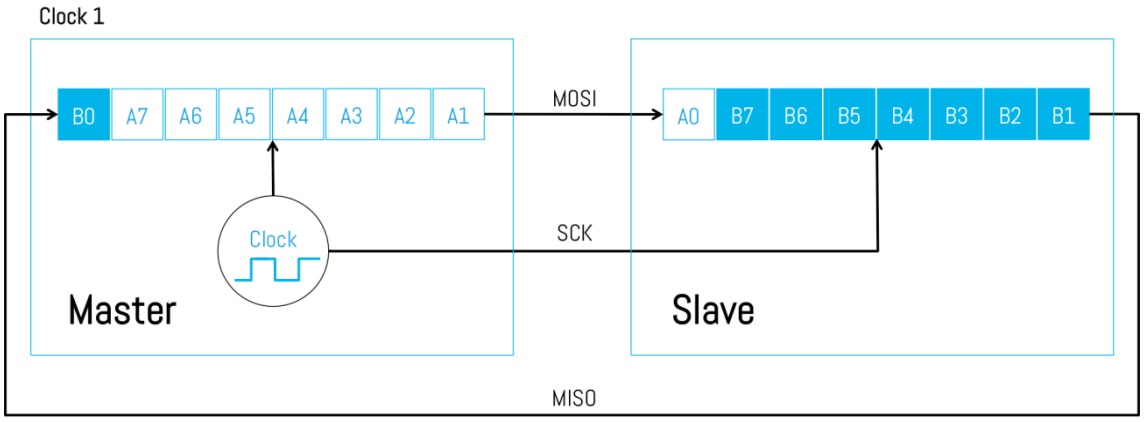


Figura 34. Transmisión del primer bit entre registros. Fuente [38]

Con cada ciclo de reloj la operación se repite, ocupando el bit menos significativo de cada registro con cada ciclo de reloj la posición del bit más significativo del registro opuesto. El resultado final es el expuesto en la figura 35, en la que después de 8 ciclos de reloj, todos los bits han sido transmitidos de un registro al otro.

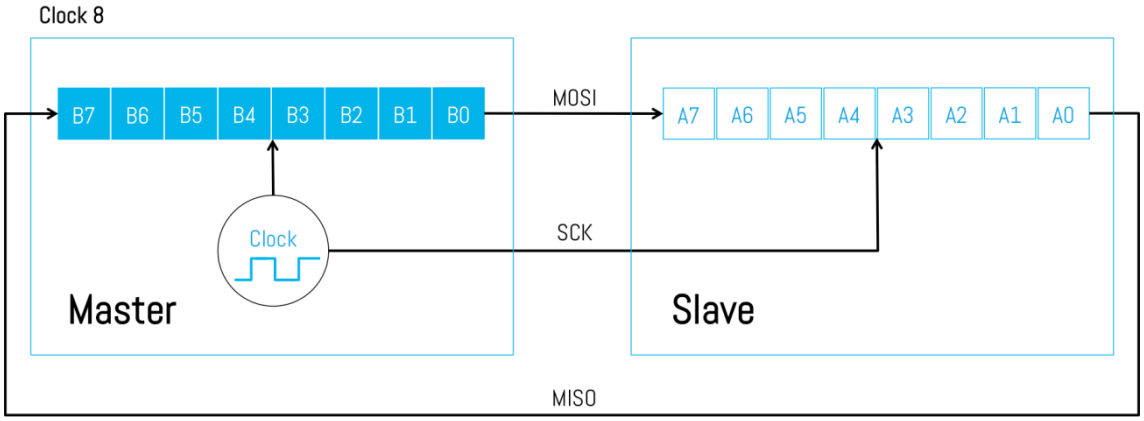


Figura 35. Transmisión completa de un byte entre registros. Fuente [38]

Arquitectura del bus SPI

En el bus SPI, el dispositivo maestro está conectado a cada dispositivo esclavo mediante 4 líneas. Cada una de estas líneas transporta una señal especifica definida por el protocolo SPI, siendo estas como se vio anteriormente: Slave select ($\overline{SS}$), Serial clock (SCK), Master out/slave in (MOSI) y Master in/slave out (MISO).

$\overline{SS}$

La primera línea, llamada *Slave Select* o $\overline{SS}$ permite al dispositivo maestro disponer de multiples esclavos. Esta línea es activa a nivel bajo, por lo que si el dispositivo maestro tiene varios dispositivos esclavos y quiere seleccionar uno de ellos, deberá enviar un valor lógico 0 a través de la línea $\overline{SS}$ que lo conecte con el esclavo con el que quiere comunicar. En la figura 36 se puede observar el modo típico de conexión entre un maestro y varios dispositivos esclavos.

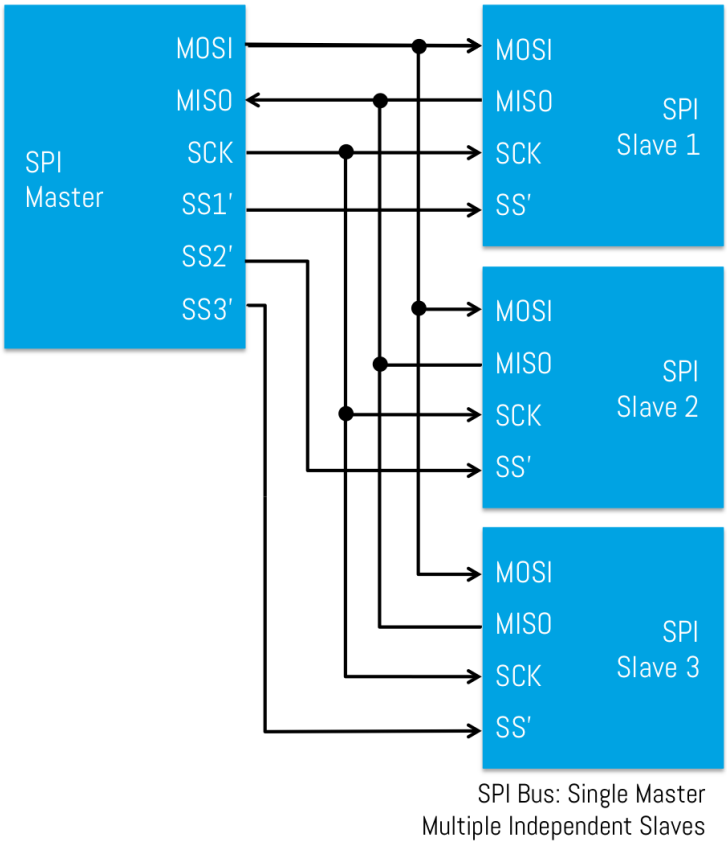


Figura 36. Dispositivo maestro conectado a múltiples dispositivos esclavos. Fuente [38]

SCK

También llamado SCLK, es el pulso que marca la sincronización en las transmisiones. Con cada pulso de este reloj, se lee o se envía un bit. El dispositivo maestro es el encargado de manejar la señal de reloj por lo tanto funciona como salida en el maestro y como entrada en los dispositivos esclavos.

MOSI

La línea MOSI *(Master Output Slave Input)*, es la encargada de conectar la salida de datos del dispositivo maestro y la entrada de datos al esclavo.

MISO

La línea MISO *(Master Input Slave Output)*, es la encargada de conectar la entrada de datos del dispositivo maestro y la salida de datos al esclavo.

Polaridad y fase de la señal de reloj

El bus SPI requiere una sincronización adicional a la otorgada por los pulsos de reloj generados por el dispositivo maestro. Es por ello que maestro y esclavo deben programarse siguiente un protocolo de sincronización especifico, el cual incluye dos nuevos conceptos: *clock polarity* (CPOL) y *clock phase* (CPHA).

Clock polarity o CPOL. Determina el valor por defecto de la línea de reloj SCK cuando el bus SPI se encuentra libre. Puede tomar dos valores: 1 o 0.

- Si CPOL = 0, el valor por defecto de la línea de reloj será 0, es decir tendrá un nivel lógico 0 cuando el bus se encuentre libre.
- Si CPOL = 1, el valor por defecto de la línea de reloj será 1, es decir tendrá un nivel lógico 1 cuando el bus se encuentre libre.

Clock Phase o CPHA. Determina cual será el flanco de la señal de reloj en el que se producirá el muestreo de datos.

- Si CPHA = 0, los datos serán muestreados en el flanco de subida.
- Si CPHA = 1, los datos serán muestreado en el flanco de bajada.

Los diferentes valores disponibles para CPOL y CPHA dan lugar a cuatro configuraciones o modos de funcionamiento del bus SPI (figura 37): Modo 0, 1, 2 o 3.

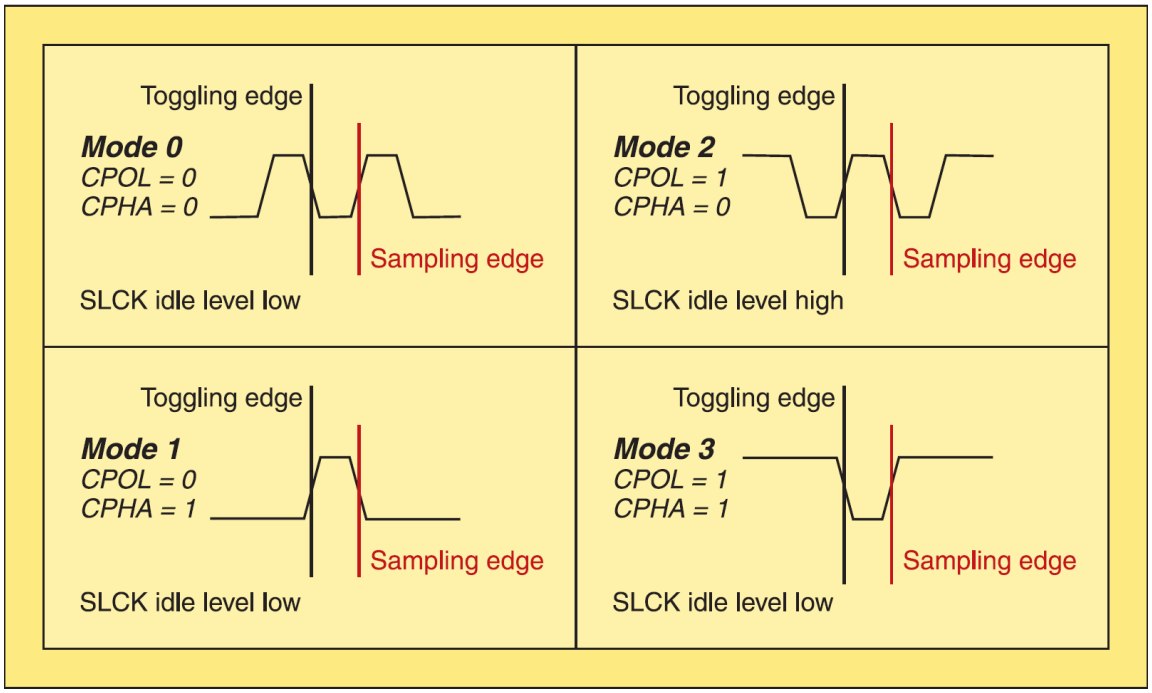


Figura 37. Modos de funcionamiento del bus SPI. Fuente [39]

4.2.2. Implementación del protocolo de comunicaciones entre microcontrolador y módulo Bluetooth nRF2740

A la hora de implementar el protocolo de comunicaciones SPI entre microcontrolador y módulo nRF2740, se explicará cómo realizar el conexionado entre ambos. Después se analizará en detalle el hardware y software utilizados y, por último, se mostrara el diseño de la aplicación software y su implementación.

Conexionado

Tomando como referencia la hoja de características del nRF2740 [40] y teniendo en cuenta que la placa de desarrollo Arduino Micro incluye soporte hardware dedicado para SPI, se realizan las conexiones necesarias (figura 38) utilizando los pines MISO, MOSI y SCK (figura 23), además de alimentar el modulo a 3.3V y conectarlo a tierra (GND). Debido a que todos los pines digitales del ATmega32u4 (incluidos los de SPI) funcionan a 5V, será necesario utilizar un conversor de voltaje bidireccional en las líneas que funcionan como salida (output) del microcontrolador hacia el nRF8001, con el fin de no dañar este último ([37] Página 32). Además, como se puede apreciar en la figura 40, existen 3 líneas adicionales: RESET, RDYN y REQN.

En lo referente a las líneas RDYN y REQN, las comunicaciones entre el chip nRF8001 incorporado en el módulo nRF2740 y el microcontrolador se realizan mediante una interfaz especial denominada ACI (*Application Controller Interface*) [37]. Esta interfaz consiste en 5 pines, 3 ya conocidos como son MISO, MOSI y SCK y dos nuevos, RDYN y REQN. Debido a que el chip no se comporta como un esclavo SPI puro, pudiendo por ejemplo recibir datos de forma inalámbrica (*over-the-air*) en cualquier momento, se utilizan estas dos señales (RDYN y REQN) a modo de señales *hand-shake* activas a nivel bajo. Cabe destacar que la línea RDYN deberá estar configurada en todo momento como entrada (*input*) al microcontrolador con resistencia de pull-up.

Esto permite que el chip pueda notificar al microcontrolador cuando ha recibido nuevos datos *over-the-air*, así como permitir al microcontrolador retrasar nuevas transmisiones de datos hasta que esté preparado para procesarlas. ([37] Página 21) Más adelante se tratará con profundidad cuando y como son utilizadas estas dos señales.

En cuanto a la línea RESET, esta simplemente permite al microcontrolador reiniciar el nRF8001 manteniendo la línea en un valor lógico bajo durante al menos 200 ns ([37] Página 48). Tanto RESET como RDYN y REQN pueden ser conectadas a cualquier pin digital IO, siendo los elegidos 9(A9), 6(A7) y 4(A6) (figura 23) respectivamente.

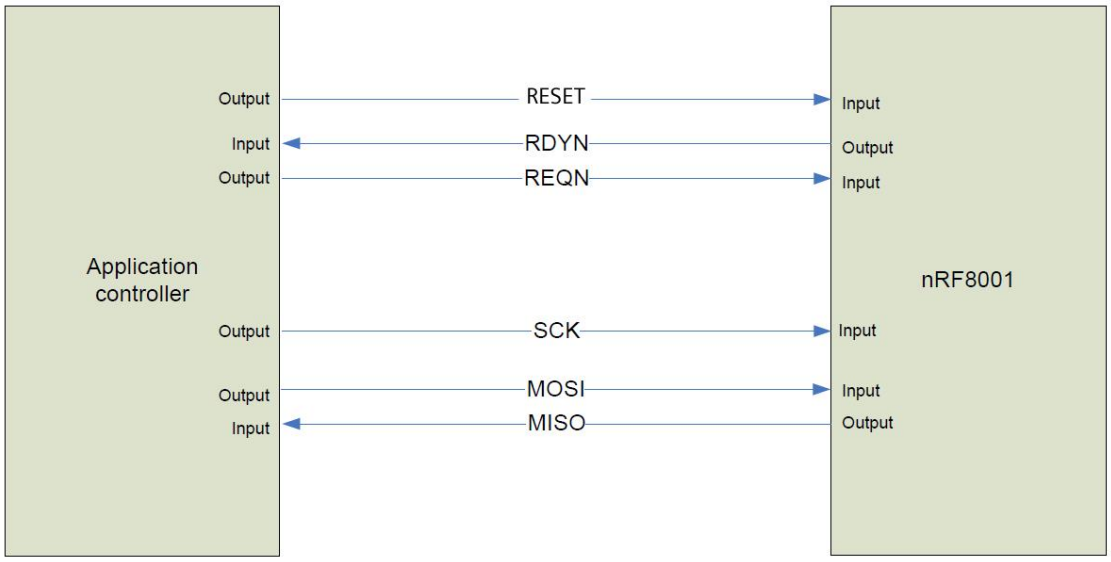


Figura 38. Conexionado de la interfaz ACI. Fuente: Nordic Semiconductor.

Soporte hardware y software en Arduino

En lo referente al soporte software, la librería SPI de Arduino permite configurar el bus SPI para que funcione acorde a lo estipulado por la hoja de características del nRF8001 ([37] Página 22). En este caso es necesario configurar el bus de la siguiente manera:

Data order	Least significant bit first
Clock polarity	Zero (base value for the clock is zero)
Clock phase	Zero (data is read on the clock's rising edge)

Table 2. SPI signal description

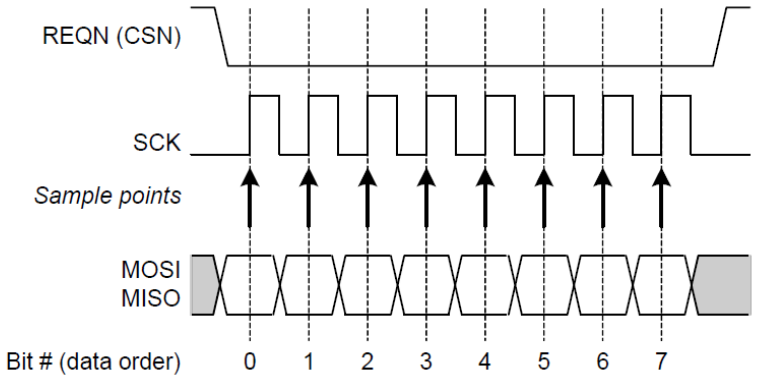


Figura 39. Configuración del bus SPI. Fuente: Nordic Semiconductor

Por un lado se observa que es necesario configurar el bus para que se transmita el bit menos significativo (LSB) primero. Esto se consigue mediante la función setBitOrder(order) de la librería SPI con order=LSBFIRST.

En cuanto a la fase y la polaridad de la señal de reloj, como se vio en el apartado anterior esta configuración corresponde al modo 0 del bus SPI (CPOL=0, CPHA=0). Utilizando la función SPI.setDataMode(mode) con mode=SPI_MODE0 se consigue el efecto deseado.

Por último se debe configurar la frecuencia a la que va a funcionar el reloj del bus SPI. Como se puede apreciar en la hoja de características del nRF8001 ([37] Página 26), el valor máximo está limitado a 3 MHz. Mediante la función SPI.setClockDivider(divider), pudiendo divider tomar un valor de 2 a 128 en potencias de 2 (2,4,8,16,32,64 y 128), es posible configurar este aspecto del bus. Como la frecuencia máxima a la que puede funcionar el reloj del microcontrolador es de 16MHz, se utilizará como argumento de

la función el código asociado a un divisor igual a 8 (SPI_CLOCK_DIV8), dando lugar a una frecuencia de reloj de 2MHz para el bus SPI.

El microcontrolador incorporado en la placa Arduino, el ATmega32u4, dispone de una unidad destinada a las comunicaciones SPI llamada *SPI UNIT.* La figura 40 muestra el diagrama de bloques de dicha unidad.

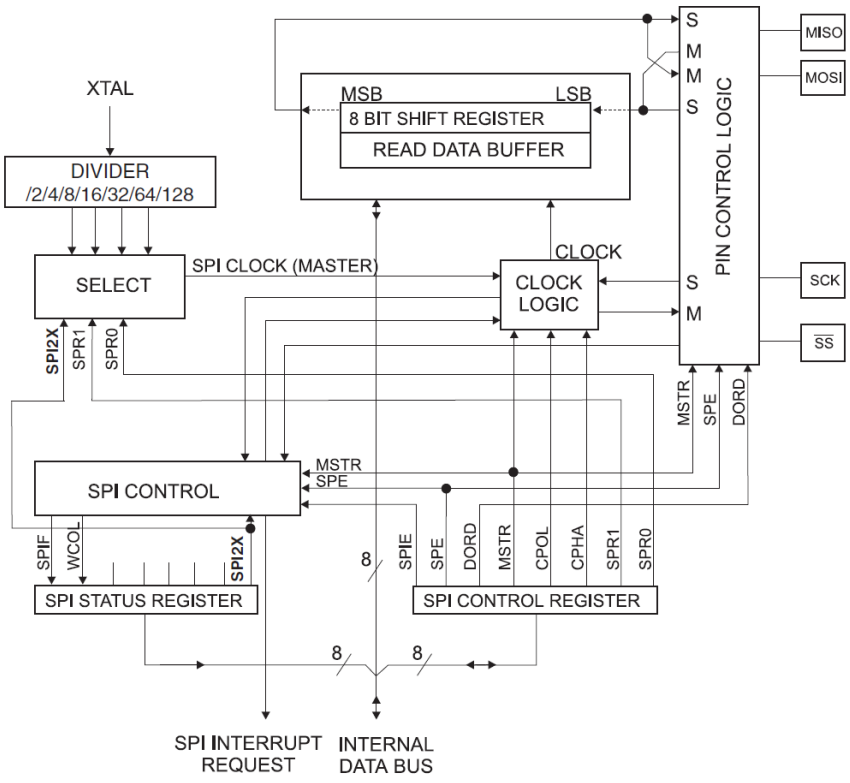


Figura 40. Diagrama de bloques SPI UNIT. Fuente ([36] Página 177)

Como se puede observar, dentro de la misma existen 3 registros diferentes: SPI STATUS REGISTER (**SPSR**), SPI CONTROL REGISTER (**SPCR**) y SPI DATA REGISTER (**SPDR**) ([36] Páginas 182-184).

Mediante el SPCR se elige si el microcontrolador funciona como dispositivo maestro o esclavo, poniendo a nivel lógico alto el bit MSTR cuando debe funcionar como maestro y a nivel lógico bajo como esclavo. Es en este registro donde tiene efecto la configuración efectuada mediante la función setBitOrder(order) que se vio con anterioridad, escribiendo un 1 en el bit DORD para que se transmita el bit menos significativo primero. También es aquí donde se producen los cambios efectuados

mediante las funciones SPI.setDataMode(mode) y SPI.setClockDivider(divider), alterando el valor de los bits CPOL, CPHA, SPR1 y SPR0.

En cuanto al registro SPSR en él se encuentran entre los bits SPIF y WCOL. El bit SPIF (*SPI Interrupt Flag*) es el bit que marca cuando se ha completado una transmisión, tomando un valor lógico 1 y generando una interrupción en el microcontrolador. El bit WCOL (*Write Collision Flag*) toma un valor lógico 1 cuando se produce una operación de escritura en el SPDR durante una transferencia de datos. Por último el registro SPDR es un registro de lectura / escritura utilizado para la transferencia de datos.

Soporte hardware y software en el módulo nRF2740

Al igual que la placa de desarrollo está basada en un microcontrolador ATmega32u4, el modulo Bluetooth nRF2740 lo está en un chip nRF8001. Con el fin de entender como se ha realizado la aplicación que realiza las comunicaciones entre el microcontrolador y el módulo nRF2740 es fundamental conocer cómo se comporta el nRF8001. En los siguientes apartados se explicara el funcionamiento del chip y como se ha configurado para el sistema diseñado.

4.2.2.1 Funcionamiento del nRF8001

Como se mencionó con anterioridad, las comunicaciones entre el chip nRF8001 y el microcontrolador son llevadas a cabo mediante una interfaz denominada ACI. Las comunicaciones mediante ACI son bidireccionales, teniendo el microcontrolador el control de las mismas. Todos los paquetes de datos enviados desde el microcontrolador hacia el nRF8001 se denominan comandos, existiendo dos tipos: comandos de sistema (*System commands*) y comandos de datos (*Data commands*).

A su vez, todos los paquetes de datos enviados desde el nRF8001 al microcontrolador son denominados eventos, existiendo dos tipos: eventos del sistema (*System events*) y eventos de datos (*data events*). En la figura 41 se puede apreciar cómo es efectuado el intercambio de comandos y eventos entre microcontrolador, nRF8001 y dispositivo remoto o *peer device.*

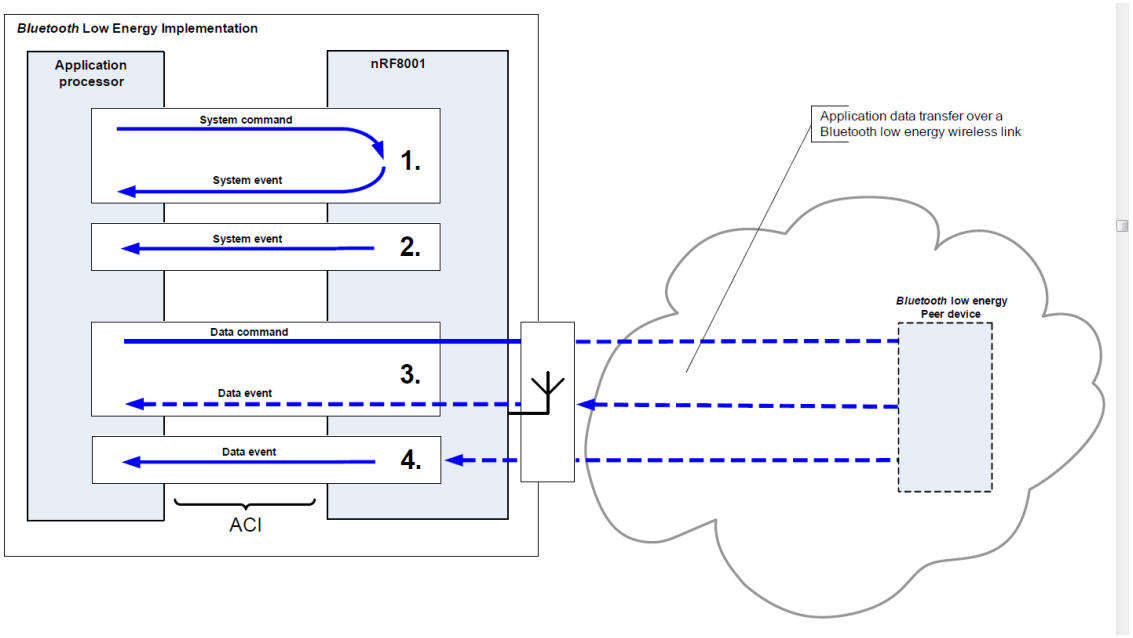


Figura 41. Principio de funcionamiento de la interfaz ACI. Fuente: Nordic Semiconductor

Tipos de paquetes

Como se pudo ver en el apartado anterior, existen varios tipos de paquetes: comandos de sistema, comandos de datos, eventos del sistema y eventos de datos.

Comandos de sistema

Los comandos de sistema son enviados por el microcontrolador para configurar, definir el modo de operación y el comportamiento del chip nRF8001. Una lista detallada de todos los comandos de sistema disponibles esta accesible en la hoja de características del nRF8001 ([37] Página 96).

Comandos de datos

Los comandos de datos o *data commands* son utilizados cuando es necesario transmitir datos entre el nRF8001 y un dispositivo conectado con el mediante BLE. Estos iniciarán la transmisión de datos cuando el nRF8001 esté actuando como servidor GATT, permitiéndole transferir datos almacenados de manera local al dispositivo remoto o recibir datos del dispositivo remoto y almacenarlos de manera local.

Si el nRF8001 actúa como cliente GATT, los comandos de datos iniciarán la transmisión de datos cuando se soliciten datos al dispositivo remoto o cuando se reciban estos datos.

A modo de apunte cabe destacar que la radio BLE del chip será la encargada de controlar el tráfico en todo momento, estando solo parcialmente atada a lo que dicten los comandos. Por ejemplo, la transmisión de datos solo ocurrirá en el intervalo de tiempo de la conexión en el que la radio este activa, sin importar si un comando acaba de entrar ordenando dicha transmisión. Una lista detallada de todos los comandos de datos disponibles esta accesible en la hoja de características del nRF8001 ([37] Página 133).

Eventos

Los eventos son enviados desde el nRF8001 al microcontrolador, pudiendo ser motivados como respuesta a un comando o generados de manera asíncrona. Estos últimos son mensajes dirigidos al microcontrolador indicando que una condición se ha cumplido. Por ejemplo, cuando la conexión RF entre el chip y un dispositivo remoto ha terminado abruptamente, generando un *DisconnectedEvent* ([37] Página 145). Una lista detallada de todos los eventos disponibles esta accesible en la hoja de características del nRF8001 ([37] Paginas 139 y 154).

Estructura de los paquetes

Todo el tráfico de información entre microcontrolador, nRF8001 y dispositivo remoto o *peer device* se organiza en paquetes. Cada paquete contiene una cabecera de 2 bytes y entre 0 y 30 bytes de carga útil, denominados PDU (*Protocol Data Unit)*, por lo que la longitud máxima de los mismos es de 32 bytes en el caso de comandos y 31 bytes para eventos. El primer byte de la cabecera contiene la longitud total del paquete en bytes. El segundo contiene el código especifico que identifica al comando / evento enviado. En cuanto a la carga útil, el número de bytes de la misma se determina según el tipo de paquete. Los bytes siguen un formato *Little Endian*, es decir el bit menos significativo es el primero que se transmite.

Envío de comandos

Una vez conocido el papel que tienen los comandos en las comunicaciones con el nRF8001 y su estructura, es hora de saber cómo se efectúa la transmisión de los mismos. En primer lugar el microcontrolador solicita al nRF8001 transmitir un paquete estableciendo un nivel lógico bajo en la línea REQN. Si el nRF8001 está preparado para recibir acepta esta transmisión poniendo la línea RDYN a nivel lógico bajo. Si la transmisión se da inmediatamente después de un reset o cuando el chip pasa de un estado dormido o *sleep* a despierto o *wake up* el nivel de la línea RDYN será válido transcurridos 62ms ([37] Página 23).

Una vez realizado el procedimiento de *hand-shake* el microcontrolador empieza a enviar el paquete a través de la línea MOSI, siguiendo la estructura explicada en la sección **Estructura de los paquetes**. Al finalizar el envío del paquete completo, el microcontrolador establece un nivel lógico alto en la línea REQN indicando que ha finalizado y el nRF8001 lo imita haciendo lo mismo con la línea RDYN.

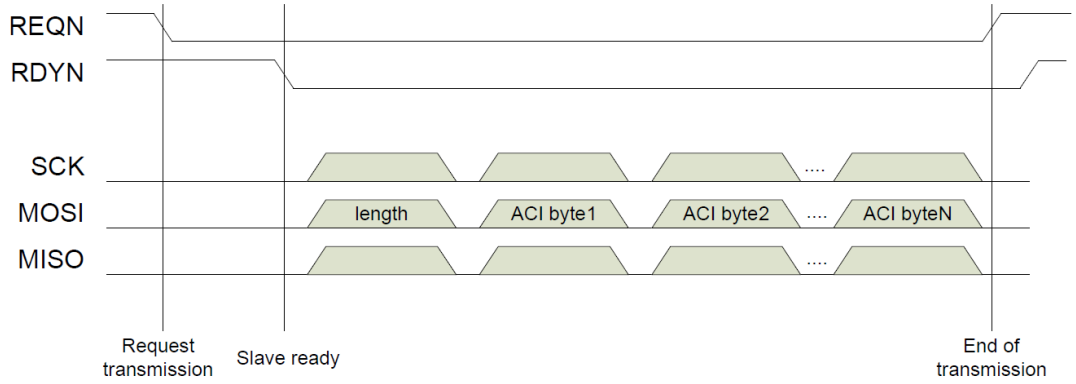


Figura 42. Intercambio de datos entre el microcontrolador y el chip nRF8001. Fuente: Nordic Semiconductor

Envío de eventos

A la hora de transmitir eventos, el chip nRF8001 solicita transmitir un paquete estableciendo un nivel lógico bajo en la línea RDYN. Si el microcontrolador está preparado para recibir acepta esta transmisión poniendo la línea REQN a nivel lógico

bajo y empieza a enviar pulsos de reloj a través de la línea SCK como mínimo 20ns después ([37] Página 26). El chip entonces comienza a enviar datos a través de la línea MISO, siguiendo la estructura explicada en la sección **Estructura de los paquetes** precedida de un byte de debug el cual es descartado por el microcontrolador. Una vez finalizada la transmisión del paquete el microcontrolador pone la línea REQN a 1 de nuevo.

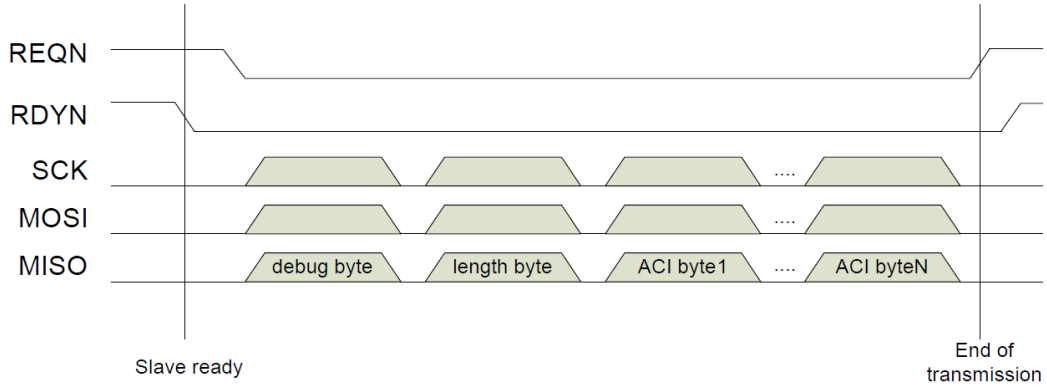


Figura 43. Recepción de un evento desde el nRF8001. Fuente: Nordic Semiconductor

Comunicación full-duplex

El chip nRF8001 es capaz de recibir un comando por parte del microcontrolador y, de manera simultánea, enviar un evento al mismo estableciendo una transmisión full-duplex en el bus SPI. Para controlar que este tipo de transmisiones se manejan de forma adecuada, el microcontrolador estará siempre a la espera de un paquete que contenga un byte de longitud superior a 0. Si esto ocurre, empezara a leer los datos transmitidos por la línea MISO según lo explicado en el apartado anterior. Cabe destacar que el evento enviado por el nRF8001 nunca será enviado como respuesta al comando recibido en ese momento. Para todos los comandos, el evento de respuesta siempre será comunicado una transmisión posterior.

Mecanismos para el control del flujo

Los comandos recibidos por el nRF8001 son ejecutados según el principio FIFO (*First In, First Out*), existiendo una diferencia en el manejo de los comandos de sistema y los comandos de datos. En lo referente a los comandos de sistema, solo uno de ellos

puede estar activo antes de que el microcontrolador pueda enviar otro. Cuando el comando pendiente haya sido ejecutado, deberá ser confirmado al microcontrolador mediante el envío de un evento o una secuencia de eventos. Todo evento enviado a modo de respuesta a un comando será enviado por el nRF8001 en un tiempo máximo de 2 segundos a partir de la recepción del comando.

En cuanto a los comandos de datos, estos son almacenados en un buffer, el cual puede almacenar en cola un número limitado de comandos. Cada comando es almacenado en un espacio del buffer, denominado crédito. Estos créditos están basados en el sistema de Bluetooth clásico de *flow control* [2] y son otorgados por el nRF8001 al microcontrolador, siendo consumidos uno a uno cada vez que este envía un comando. Gracias a este sistema el nRF8001 puede controlar el flujo de comandos que es capaz de gestionar, enviando un evento indicando que no tiene créditos disponibles si el microcontrolador sobrepasa dicha capacidad.

Deben cumplirse dos condiciones antes del envió de cualquier comando por parte del microcontrolador. En primer lugar un evento del tipo *DeviceStartedEvent* ([37] Página 139) debe ser recibido por parte del microcontrolador, el cual es enviado cuando el nRF8001 ha cambiado de modo de operación o ha sido reiniciado. Este evento contiene información relativa al modo de operación en el que se encuentra el nRF8001, el número de créditos disponibles y la existencia o no de algún error de tipo hardware, siendo de vital importancia conocer estos datos antes del envío de cualquier comando por parte del microcontrolador.

En segundo lugar, las *service pipes* (ver sección **Service Pipes**) utilizadas en la aplicación deben estar abiertas. La confirmación del estado de las pipes es posible mediante la recepción de un evento del tipo *PipeStatusEvent* ([37] Página 148), el cual contiene información sobre el estado de las mismas y su disponibilidad. Gracias a esta comprobación se asegura el correcto funcionamiento del envío de datos hacia el dispositivo remoto, garantizando que la *service pipe* utilizada para enviar información esté operativa.

Modos de operación del nRF8001

El chip nRF8001 dispone de 4 modos diferentes de operación: Sleep, Setup, Active y Test (figura 44).

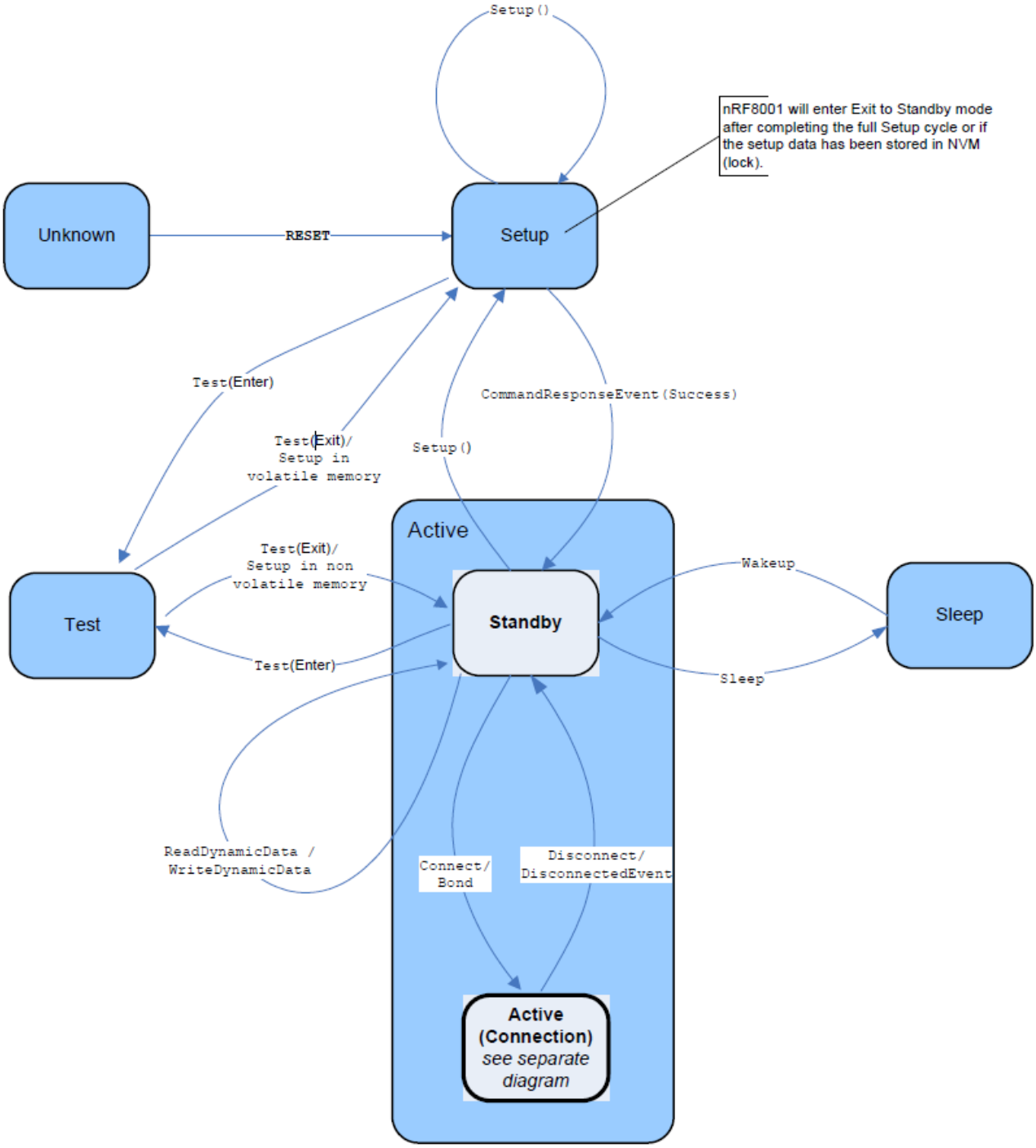


Figura 44. Máquina de estados de los modos de operación del chip nRF8001. Fuente: Nordic Semiconductor

De estos 4 modos el sistema diseñado utilizará 3: Setup, Active y Sleep. Por defecto el nRF8001 se encuentra en modo Setup, siendo controlada la transición entre modos mediante comandos de sistema enviados por el microcontrolador, como puede apreciarse en la figura 44.

Cada modo permite el uso de diferentes comandos y eventos, restringiendo el uso de los mismos a un modo de operación determinado o a varios.

Modo Setup

El modo de configuración o *Setup mode* permite configurar el nRF8001 a partir de unos archivos de configuración generados mediante una herramienta software propiedad de Nordic llamada nRFgo Studio. La configuración puede estar almacenada en la memoria volátil (RAM) o en la memoria no volátil del chip. En el primer caso el nRF8001 deberá ser configurado cada vez que sea reiniciado o deje de ser alimentado mediante una fuente de voltaje o de forma externa. En el segundo caso la configuración será bloqueada una vez escrita en la memoria no volátil y no podrá ser reprogramada. La configuración del nRF8001 incluye:

-Ajustes relativos al perfil GAP: Definen el comportamiento del chip en el modo operativo normal (Active), además de definir otros parámetros específicos de BLE como el nombre del dispositivo o como realiza el procedimiento de *advertising*.

-Ajustes relativos al perfil GATT: Definen los servicios (y las características asociadas a ellos) que el chip va a soportar, así como las *service pipes* utilizadas en la aplicación (ver sección **Service Pipes**).

-Ajustes hardware: Definen los ajustes relativos a elementos hardware del nRF8001 como la radio, antena EIRP, convertidor DC/DC, etc.

Una vez realizada la configuración mediante el software nRFgo Studio esta podrá ser cargada en el nRF8001 mediante el envío de una batería de comandos de sistema de tipo *Setup* ([37] Página 102), los cuales se encuentran en los archivos generados por el nRFgo Studio. En el apartado **Configuración del nRF8001** se explicará paso a paso como se ha realizado este proceso en el sistema diseñado.

El procedimiento de configuración del nRF8001 deberá ser completado antes de poder utilizar el chip para enviar o recibir datos vía Bluetooth Low Energy.

Modo Sleep

El modo Sleep o *Sleep mode* permite ahorrar batería al chip nRF8001 cuando este no se encuentra conectado a un dispositivo remoto o realizando *advertising*. Cuando el chip se encuentra en este modo todas las conexiones son desactivadas y los ajustes de configuración guardados, permitiendo retomar las operaciones sin tener que configurar de nuevo el sistema en el caso en el que la configuración haya sido almacenada en la memoria volátil. Como se puede ver en la figura 44, mediante el comando de sistema *Sleep* el nRF8001 entra en este modo, saliendo del mediante el comando *Wakeup* ([37] Páginas 100 y 101).

Modo Active

El modo activo o *Active mode* del chip nRF8001 es el encargado de las operación en tiempo de ejecución y el intercambio de datos de aplicación. El chip nRF8001 puede entrar en modo activo una vez finalizado el proceso de configuración, cuando recibe un comando *Wakeup* estando en modo *Sleep* o desde un reset si la configuración había sido guardada en la memoria no volátil. El modo activo permite acceder a la mayoría de funcionalidades del chip nRF8001.

Dentro del modo activo existen diferentes estados o niveles de actividad: Standby, Advertising y Connected (figura 45).

Estado Standby

El estado Standby es el nivel de actividad inicial en el que se encuentra el nRF8001 al entrar en el modo Activo. En este estado no existe actividad de la radio ni de cualquier otro elemento relacionado de manera activa con las comunicaciones vía Bluetooth low energy.

Estado Advertising

Las transiciones desde el estado Standby al estado Advertising se producen cuando el nRF8001 recibe un comando *Connect* o *Bond* ([37] Páginas 111 y 113) con el fin de establecer una conexión a un dispositivo remoto en rol Central. Dependiendo del tipo de comando y de la configuración del nRF8001 este podrá realizar *advertising* en uno de los tres modos disponibles:

• General Discoverable mode [41]

• Limited Discoverable mode [42]

• Non-Discoverable mode [43]

En todos los modos se establecerá un valor de *timeout*, por lo que el nRF8001 realizará *advertising* hasta que establezca conexión con un dispositivo remoto o hasta que se supere el valor de dicho *timeout*.

En el caso de que se establezca con éxito una conexión, el chip nRF8001 comenzará de inmediato el procedimiento de *Service Discovery.* Gracias al *Service Discovery* el nRF8001 descubre si los servicios Bluetooth que utilizará la aplicación, definidos en el procedimiento de configuración del chip, están disponibles en el dispositivo remoto. Para tal fin, el procedimiento de *Service Discovery* activa una serie de procedimientos asociados al perfil GATT:

• Discover Primary Services by Service UUID [44]

• Discover All Characteristics of a Service[45]

• Discover All Characteristic Descriptors[46]

• Enabling the Service Changed characteristic[47]

Es posible que el nRF8001 tenga que reactivar el procedimiento *Service Discovery* durante una conexión activa si algún servicio Bluetooth cambia durante la misma o si reconecta con un dispositivo que contiene la característica Bluetooth *Service Changed*. Una vez finalizado con éxito el procedimiento de *Service Discovery,* el nRF8001

devolverá un *ConnectedEvent* ([37] Página 143) seguido de un *PipeStatusEvent*, indicando la transición a un estado Connected.

Estado Connected

En este estado el nRF8001 ya está preparado para cualquier intercambio de datos con el dispositivo remoto. La conexión permanecerá activa hasta que el microcontrolador o el dispositivo remoto deseen finalizarla, hasta que el dispositivo remoto salga del alcance del nRF8001 o hasta que se produzca un fallo en el sistema que impida mantener la conexión. Si por cualquier causa el nRF8001 se desconecta del dispositivo remoto, volverá al estado Standby hasta que un nuevo comando *Connect* o *Bond* sea enviado por el microcontrolador.

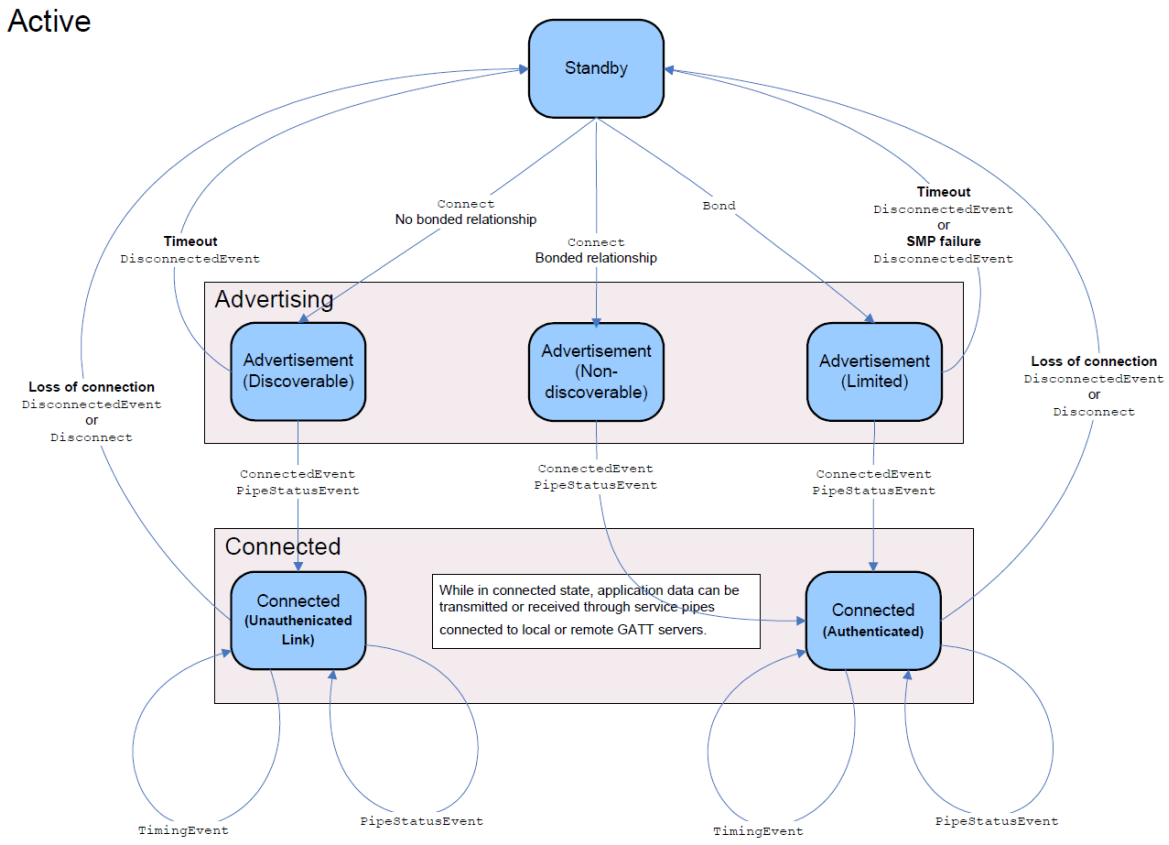


Figura 45. Máquina de estados del modo Active. Fuente: Nordic Semiconductor

Bonding

Anteriormente se ha visto como el comando *Bond*, al igual que el comando *Connect*, inicia la transición del estado Standy al estado Advertising, pero, ¿qué diferencia existe entre ambos comandos y en que consiste el proceso de *bonding*?

El proceso de *bonding* es el proceso mediante el cual el chip nRF8001 y un dispositivo remoto intercambian las claves de seguridad (ver apartado **La seguridad en BLE)** e información relativa al modo de asociación según lo descrito en la especificación Bluetooth low energy [2]. Este proceso solo es necesario si la aplicación BLE requiere un mecanismo de autentificación entre ambas partes, no siendo necesario para establecer una conexión "no autentificada" entre nRF8001 y dispositivo remoto, la cual se lleva a cabo utilizando el comando *Connect*.

Si se quiere utilizar un modo de asociación en el proceso de *bonding* de tipo PASSKEY (ver apartado **La seguridad en BLE)** con el fin de obtener protección ante un eventual ataque de tipo *Man In The Middle* (MITM), un evento del tipo *DisplayPasskeyEvent* o *KeyRequestEvent* será generado ([37] Páginas 152 y 153). Cuando el microcontrolador reciba el evento *KeyRequestEvent* deberá responder con un comando *SetKey* ([37] Página 124).

Service pipes

Si hay algo que diferencia al nRF8001 sobre otros chips BLE, es la implementación de un nuevo concepto ajeno a la especificación Bluetooth, las *service pipes*. Un aspecto clave de las comunicaciones mediante BLE es el acceso a diferentes características de un servicio Bluetooth tanto desde el dispositivo que actúa como servidor como del cliente. Por ejemplo, la característica *Temperature measurement* del servicio *Health Thermometer Service,* la cual se utilizará para transmitir el valor de la temperatura medida (ver apartado **Configuración del nRF8001**). Bien, con el fin de facilitar este acceso, Nordic crea estas "pipes" las cuales "encapsulan" los datos, obligando a que toda transmisión de los mismos se efectúe a través de ellas.

Además, es posible definir otros parámetros aparte de que característica se quiere transmitir en cada *service pipe* como por ejemplo la dirección de la transferencia,

donde se almacena el valor de la misma, si se requiere encriptación para realizar una transmisión del valor, etc. En el sistema se definirá una *service pipe* de tipo *Transmit with ACK pipe*. En el apartado **Configuración del nRF8001** se explicará en detalle cómo se realiza el proceso de configuración de las *services pipes* utilizadas por el sistema.

Creando Service pipes

A la hora de definir las *service pipes* que se van a necesitar en el sistema es necesario primero definir los servicios a los que estarán asociadas dentro del servidor GATT local (el nRF8001 en este caso). Una vez definidas se programarán mediante el programa nRFgo Studio, software propiedad de Nordic que nos permite configurar diferentes aspectos funcionales del nRF8001.

Este programa asignará un identificador numérico a cada *pipe y* lo asociará al UUID de la característica especificada. Después, generará una serie de archivos que deberán ser cargados en el chip nRF8001 mediante el envío de comandos de sistema de tipo *Setup*. Estos comandos deberán ser enviados cuando el nRF8001 se encuentre en modo Setup. Tanto el proceso de configuración del nRF8001 como el uso del software nRFgo Studio serán explicados en detalle en el apartado **Configuración del nRF8001**.

Cualquier *service pipe* que vaya a ser usada por la aplicación debe encontrarse abierta antes de que tenga lugar una transferencia de datos. La disponibilidad de cada *service pipe* viene determinada por el evento *PipeStatusEvent*. Dicho evento contiene dos listas en forma de *bitmaps*:

• Pipes Open Bitmap: Lista en la cual se indican las *service pipes* abiertas en ese momento, a través de las cuales se podrán recibir o transmitir datos.

• Pipes Closed Bitmap: Lista en la cual se indican las *service pipes* cerradas en ese momento, las cuales precisaran de un comando de tipo *OpenRemotePipe* ([37] Página 120) por parte del microcontrolador para poder ser utilizadas.

En cuanto se produzca un cambio en la disponibilidad de cualquiera de las *service pipes* utilizadas por la aplicación, un nuevo *PipeStatusEvent* será enviado al microcontrolador indicando el estado de las mismas.

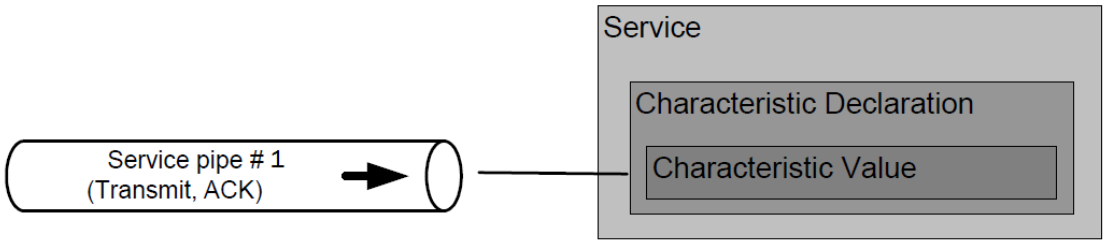


Figura 46. Ejemplo de *transmit pipe with ACK*. Fuente: Nordic Semiconductor

Transmit service pipes

Siguiendo con el ejemplo visto en la figura 46, para poder enviar información a través de la pipe 1 esta deberá estar configurada como una *transmit pipe*. Las *transmit pipes* permiten enviar datos de la aplicación a un dispositivo remoto, existiendo dos tipos:

- *Transmit pipes.*
- *Transmit pipes with acknowledge*.

Cada *transmit pipe* estará asociada con una característica de un servicio Bluetooth. En las *transmit pipes with acknowledge,* al enviar datos de la aplicación al dispositivo remoto este deberá responder con un ACK al nRF8001, indicando que ha recibido los datos. La recepción del ACK generará un *DataAckEvent* ([37] Página 157). Como se verá en el apartado **Configuración del nRF8001**, la *transmit pipe with ACK* será la utilizada para transmitir el valor de la temperatura medida por el sensor. En la figura 47 se puede observar cómo se produce la transferencia de datos en una *transmit pipe with ACK.*

Transferencia de datos mediante una service pipe

Una vez configurado el chip nRF8001, encontrándose en modo activo (ver apartado **Modos de operación del nRF8001**), el sistema enviará en algún momento información al dispositivo remoto. Esta transmisión se producirá mediante el envío de un comando

de datos como puede apreciarse en la figura 47 (por ejemplo *SendData* [37] Página 135).

Como se vio anteriormente, cada pipe tiene asignado un identificador numérico, por lo que si se quiere enviar información a través de la *pipe* asociada a la característica *Temperature measurement* la cual tiene un identificador numérico=1, el comando enviado será *SendData*(1, 0x21), el cual enviara a través de la *pipe* 1 0x21. Para transmitir o recibir información a través de una *pipe* esta deberá encontrarse abierta y el nRF8001 deberá estar conectado al dispositivo al que se quiera enviar la información.

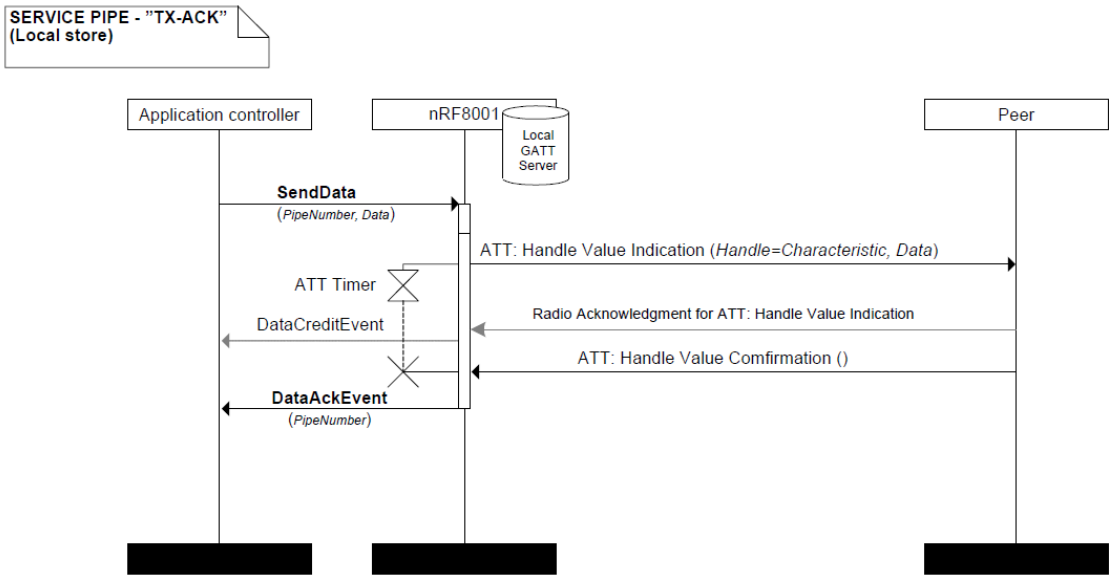


Figura 47. Transferencia de datos en una *Transmit pipes with acknowledge.* Fuente: Nordic Semiconductor.

4.2.2.2 Configuración del nRF8001

Una vez comprendido el funcionamiento del chip nRF8001 y antes de comenzar a programar la aplicación software del sistema, es necesario configurar el nRF8001. Como se explicó en la sección **Modo Setup**, el chip se configura a partir de unos archivos generados mediante una herramienta software propiedad de Nordic llamada nRFgo Studio.

Generación de los archivos de configuración mediante nRFgo Studio

Una vez instalado el programa disponible de manera gratuita en la web de Nordic (http://www.nordicsemi.com/eng/Products/2.4GHz-RF/nRFgo-Studio), se abre el programa y se genera un nuevo archivo .xml mediante File -> New -> nRF8001, escogiendo la versión de chip DX / D. Existe otra versión del nRF8001 denominada CX / C pero corresponde a las versiones más antiguas del chip, estando descatalogada en la mayoría de proveedores a nivel internacional. El resultado deberá ser parecido al mostrado en la figura 48.

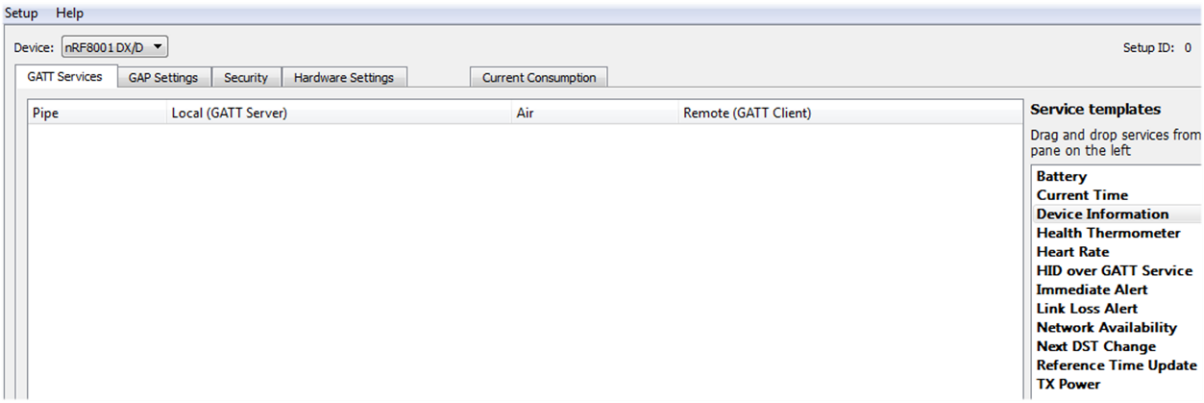


Figura 48. nRFgo Studio: Pantalla principal de configuración.

Pestaña GATT services

En la pantalla inicial, dentro de la pestaña GATT services, se añadirán los servicios Bluetooth que va a utilizar la aplicación del sistema. A primera vista el sistema se ajusta perfectamente al perfil Bluetooth HEALTH THERMOMETER [49], ya que este es utilizado cuando se dispone de un dispositivo que recolecta datos de un sensor de temperatura, definiendo dos roles: Termómetro y recolector. Sin embargo, este perfil incluye un servicio Bluetooh que no es necesario para el sistema diseñado, por lo que se utilizará una versión adaptada del mismo a los requerimientos del sistema como se verá a continuación.

Este perfil Bluetooth define que el termómetro actúe como servidor GATT y el recolector como cliente GATT, es decir, que el servidor GATT dispondrá de los datos de la aplicación y el cliente solicitará al servidor tener acceso a esos datos.

En el sistema diseñado el termómetro no es un dispositivo Bluetooth, sino que es un sensor de temperatura digital que se comunica con el nRF8001 mediante un microcontrolador. Esto significa que el que debería ser el servidor GATT (el termómetro) no puede actuar como tal al no disponer de conectividad Bluetooth, actuando como servidor GATT el chip nRF8001 en su lugar. Por otro lado, el recolector será siempre un dispositivo móvil (en este caso un smartphone o tableta), el cual actuará como cliente GATT enviando peticiones al servidor para tener acceso a las mediciones del sensor de temperatura.

El perfil Bluetooth HEALTH THERMOMETER define una serie de servicios que deben incluirse para acreditar que se está utilizando ese perfil y no otro, en este caso son dos: el servicio Health Thermometer [50] y el servicio Device Information.

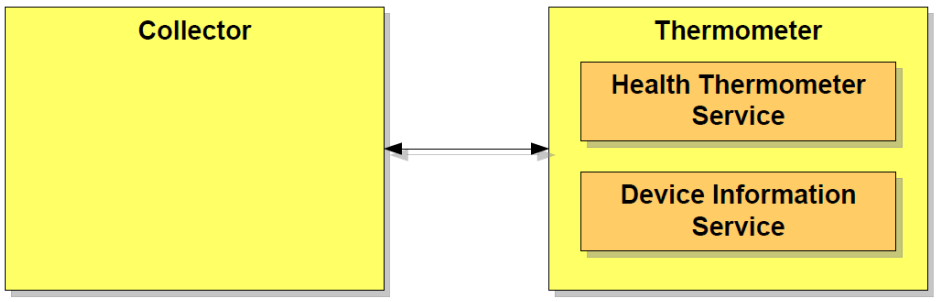


Figura 49. Roles y servicios definidos en el HEALTH THERMOMETER PROFILE. Fuente [49]

Como se ha comentado, este perfil incluye un servicio Bluetooth que carece de utilidad en el sistema diseñado, en concreto el servicio *Device Information*, el cual tiene asociadas características que se utilizan para almacenar información sobre el fabricante del chip, el proveedor, etc. Por tanto, solo se utilizará el servicio *Health Thermometer*, excluyendo el servicio *Device Information* de la configuración del chip.

Para configurar los servicios Bluetooth que se utilizarán en la aplicación del sistema se deben arrastrar los servicios indicados desde la lista denominada “Service templates” hasta el espacio en blanco ubicado debajo de “Local (GATT server)” (figura 50) y, posteriormente, elegir qué características de las disponibles en el servicio elegido se utilizarán. En el caso del servicio *Health Thermometer*, se seleccionará la característica

Temperature Measurement. En la figura 50 se puede apreciar como el programa marcar el servicio con una “S” y la característica con una “C”.

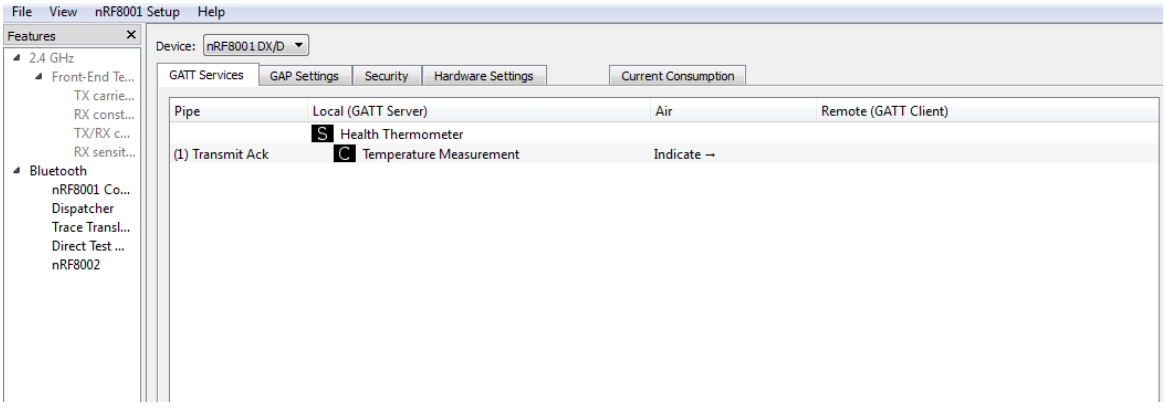


Figura 50. Elección de los servicios y características de la aplicación.

Una vez añadido el servicio haciendo click sobre él se observa lo siguiente (figura 51).

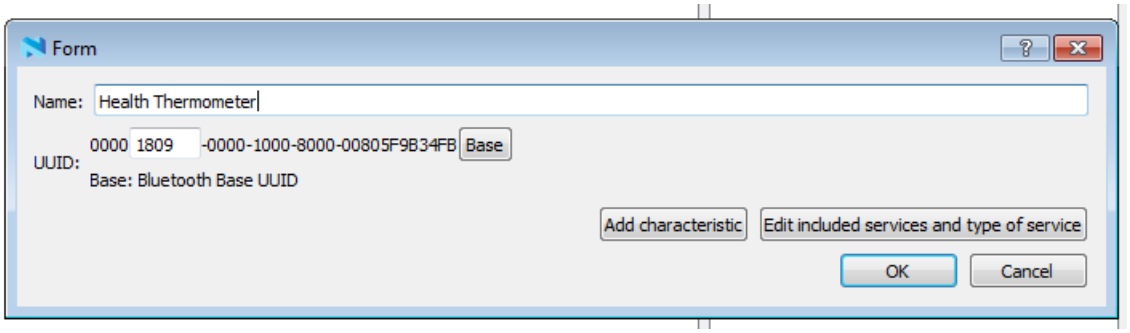


Figura 51. Detalles del servicio *Health Thermometer*.

En la figura 51 se puede apreciar un campo denominado UUID seguido de 16 bytes. Las siglas UUID responden a *Universally Unique Identifier*, un identificador de 128 bits único para cada servicio y característica Bluetooth. En este caso como el sistema utiliza servicios y características ya definidos por la especificación Bluetooth, el UUID viene predefinido por defecto. Como se puede apreciar en la figura 51 solo es posible cambiar 2 bytes, estos bytes son conocidos como *Short UUID form* y representan la forma reducida (16 bits en vez de 128) del identificador UUID.

Así, la característica Temperature Measurement también muestra su UUID acorde con la especificación Bluetooth [51].

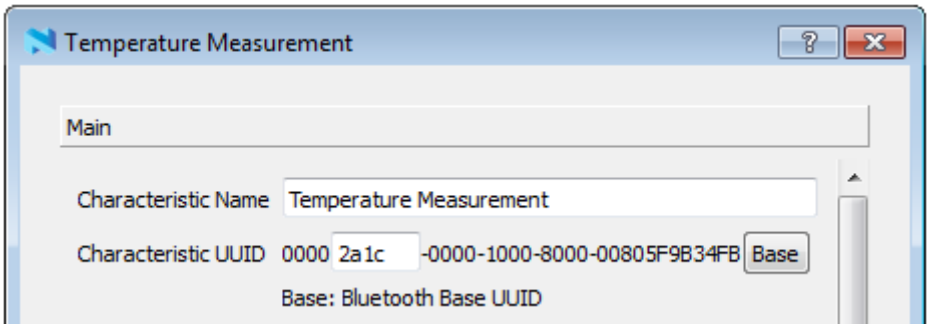


Figura 52. Detalles de la caracteristica *Temperature Measurement*.

En la figura 50 se puede apreciar como en la pestaña GATT services existe un campo llamado Pipe, como se comentó en el apartado *service pipes,* el sistema diseñado utilizará una, en este caso asociada a la característica *Temperature Measurement* del servicio *Health Thermometer*.

En el apartado *Properties* de la característica (figura 53) se definirán las propiedades de la misma. Las propiedades de las características pertenecientes a un servicio Bluetooth están accesibles en la especificación Bluetooth [52]. Como el sistema diseñado enviará medidas de temperatura de forma periódica una vez se haya establecido una conexión estable, se configurara la característica como “Indicate”. Una característica configurada como “Indicate” permite realizar operaciones con esa característica habilitadas por el cliente pero iniciadas por el servidor, proporcionando una manera de “empujar” la información al dispositivo cliente. Ademes, todas las características configuradas como “Indicate” son *acknowledged*, es decir, una vez el cliente ha recibido el valor de la característica debe notificar al servidor su recepción. Esto incrementa la fiabilidad del sistema a costa de sacrificar un poco de velocidad, lo cual no es un factor determinante ya que la temperatura no es una magnitud que varíe de forma extrema bajo condiciones normales de funcionamiento y saber que el dispositivo remoto ha recibido la medición sí.

Por tanto, la *pipe* vinculada con esta característica será del tipo *Transmit with ACK*, ya que se utilizara para transmitir el valor de la característica desde el servidor y deberá recibir un ACK desde el cliente.

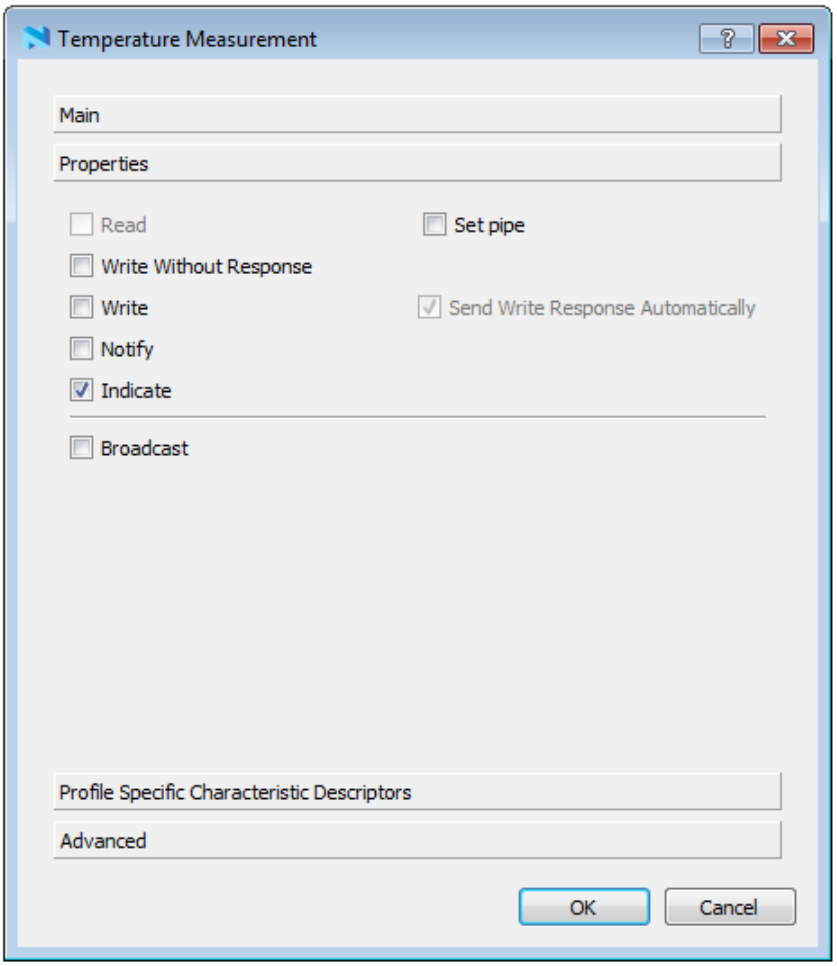


Figura 53. Propiedades de la característica *Temperature Measurement*.

Pestaña GAP settings

En esta pestaña se podrán configurar entre otros parámetros el nombre que verán los dispositivos remotos que intenten conectar con el nRF8001 (figura 54, Bluetooth Device Name). Esto será posible en el apartado Setup -> General. En este caso como ejemplo se puede ver el nombre “Termómetro Bluetooth”. Existen otras opciones para definir el nombre ya sea “Over the air” mediante el dispositivo remoto o desde el microcontrolador seleccionando la opción “From application controller (with pipe)” la cual creara una *service pipe* llamada PIPE_GAP_DEVICE_NAME_SET que permitirá al microcontrolador cambiar el nombre en tiempo de ejecución.

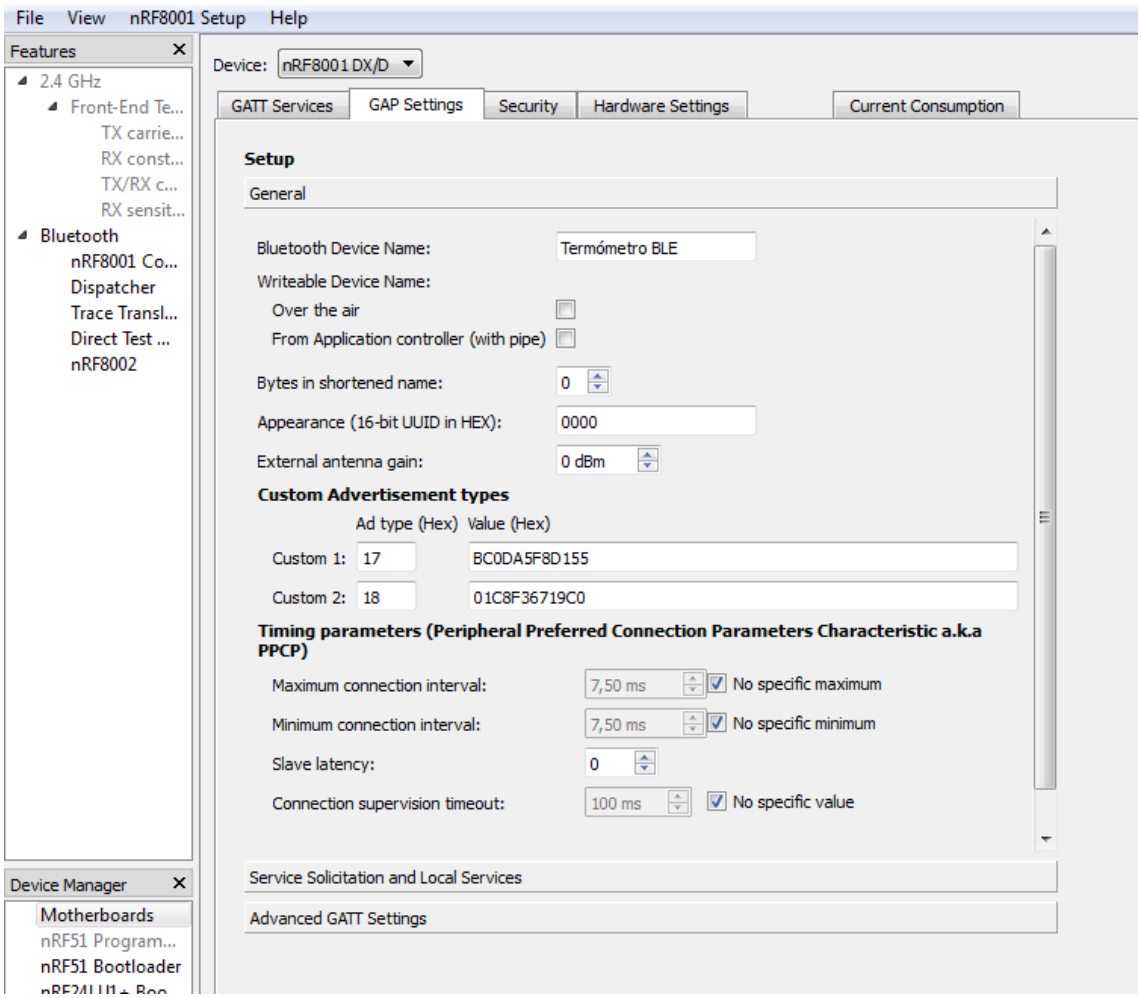


Figura 54. Pestaña GAP Settings

En el apartado Setup -> Service Solicitation and Local Services deberán definirse los servicios que el servidor GATT transmitirá mediante *advertising*, en este caso *Health Thermometer*, así como los servicios a solicitar en el cliente GATT, ninguno en este caso (figura 55).

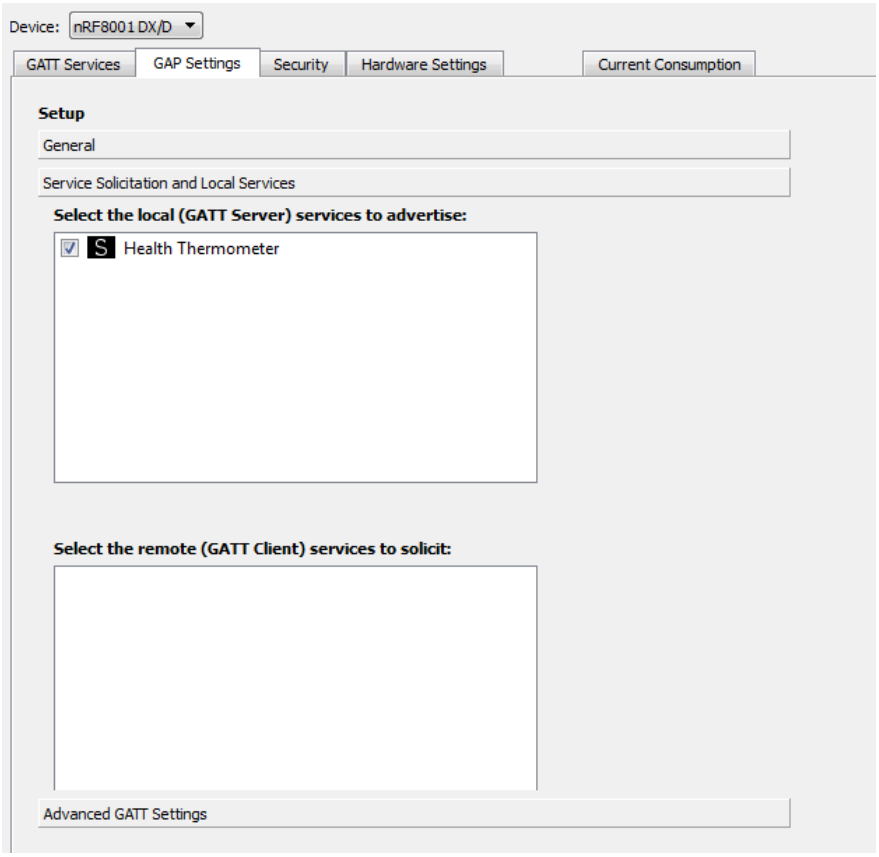


Figura 55. Service Solicitation and Local Services.

En cuanto al apartado Setup -> Advanced GATT settings, este se dejara con los valores por defecto. A la derecha, bajo el nombre "Selected fields to advertise" se podrá seleccionar que contendrá el paquete de *advertising* que enviará el nRF8001 mediante los diferentes comandos recibidos así como si incluirá el nombre antes configurado o no y como (completo o parcial). En este caso se incluirá el nombre completo y los servicios disponibles del servidor GATT (el nRF8001) en el paquete, tanto cuando las comunicaciones se inician con un comando tipo *Connect* (figura 56) como tipo *Bond* (figura 57).

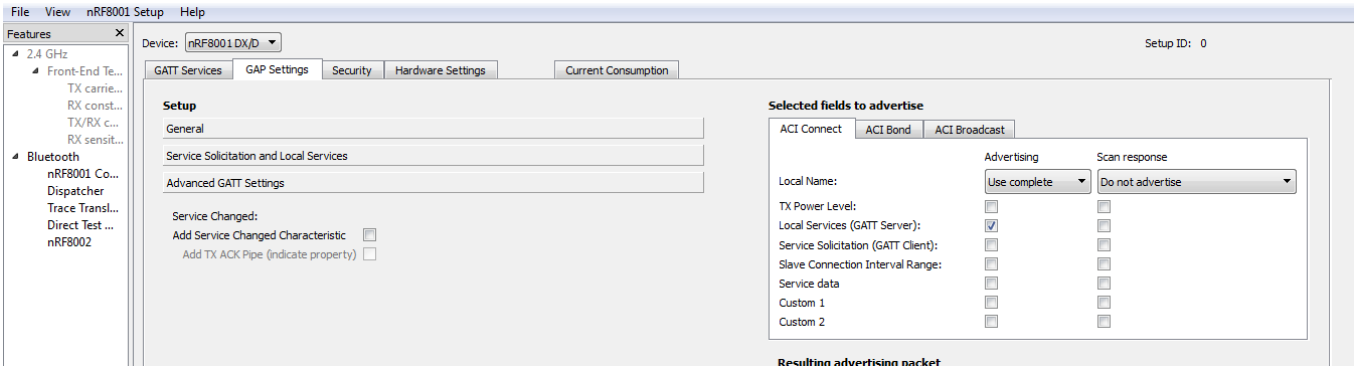


Figura 56. Selected fields to advertise (Connect).

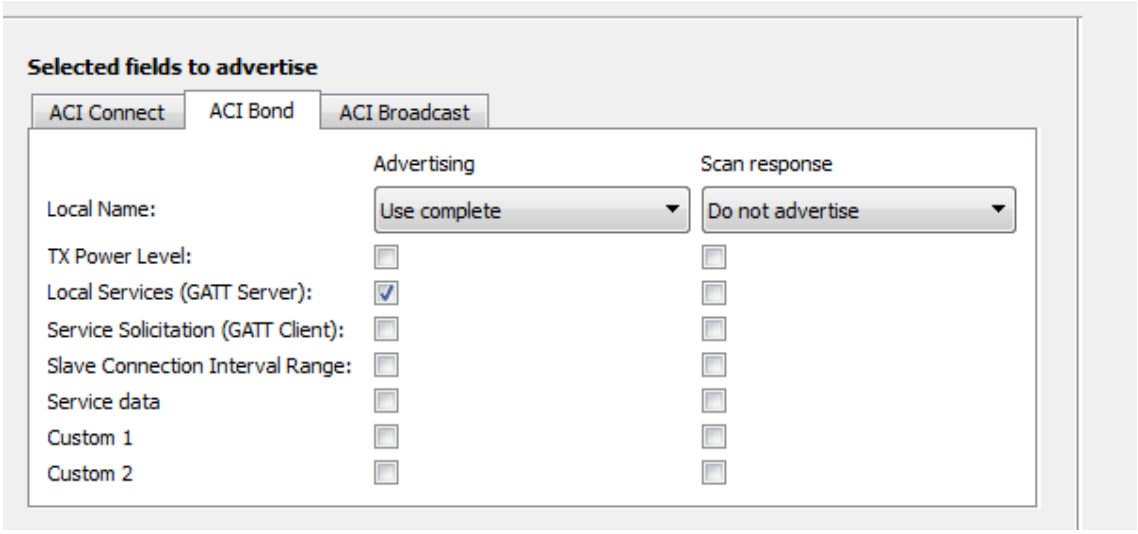


Figura 57. Selected fields to advertise (Bond).

Pestaña Security

En esta pestaña (figura 58) se podrá configurar el nivel de seguridad y los ajustes asociados a el de la aplicación Bluetooth diseñada (ver apartado **Seguridad en BLE**). En este caso se seleccionara "No security required" lo que permitiría que los datos de la aplicación sean accesibles desde el dispositivo maestro (el smartphone / tableta en este caso) sin requerir un procedimiento de *Bonding* primero. Si fuera seleccionado "Security Required" los datos de la aplicación solo serían accesibles si la conexión entre nRF8001 y dispositivo remoto se realizase a partir de un comando *Bond* en vez de un comando *Connect*.

En cuanto al apartado "Required level of security", este se refiere al modelo de asociación bajo el cual se realizará el *pairing* entre dispositivo maestro y esclavo. Se ha escogido el modo "Just Works" ya que es el modo que presenta una mayor compatibilidad con cualquier dispositivo remoto. La razón de esta elección es que si el dispositivo maestro utilizado no soporta el nivel de seguridad configurado, no será posible establecer una conexión y el sistema no cumplirá su función.

En cuanto al apartado "I/O capabilities**"**, este solo será de utilidad cuando se use un modelo de asociación que ofrezca protección ante un ataque de tipo MITM, como por ejemplo *Passkey Entry* (ver apartado **Seguridad en BLE**), el cual necesita poder mostrar y procesar datos para realizar el *pairing.* Los otros dos ajustes, "Bond timeout" y

"Security request delay" solo serán de utilidad cuando se haya seleccionado "Security Required" en el campo de "Device Security".

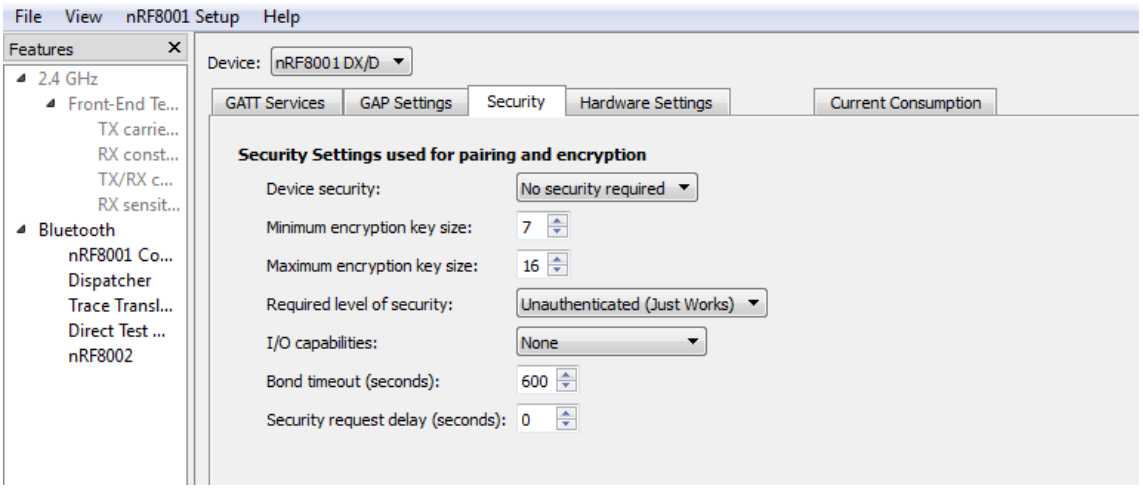


Figura 58. Pestaña Security

Pestaña Hardware Settings

En esta pestaña (figura 59) será posible realizar los ajustes relativos al hardware del nRF8001. En el caso del sistema diseñado, como se utiliza un módulo nRF2740 el cual está diseñado específicamente para funcionar con el nRF8001, no será necesario alterar los valores por defecto [53].

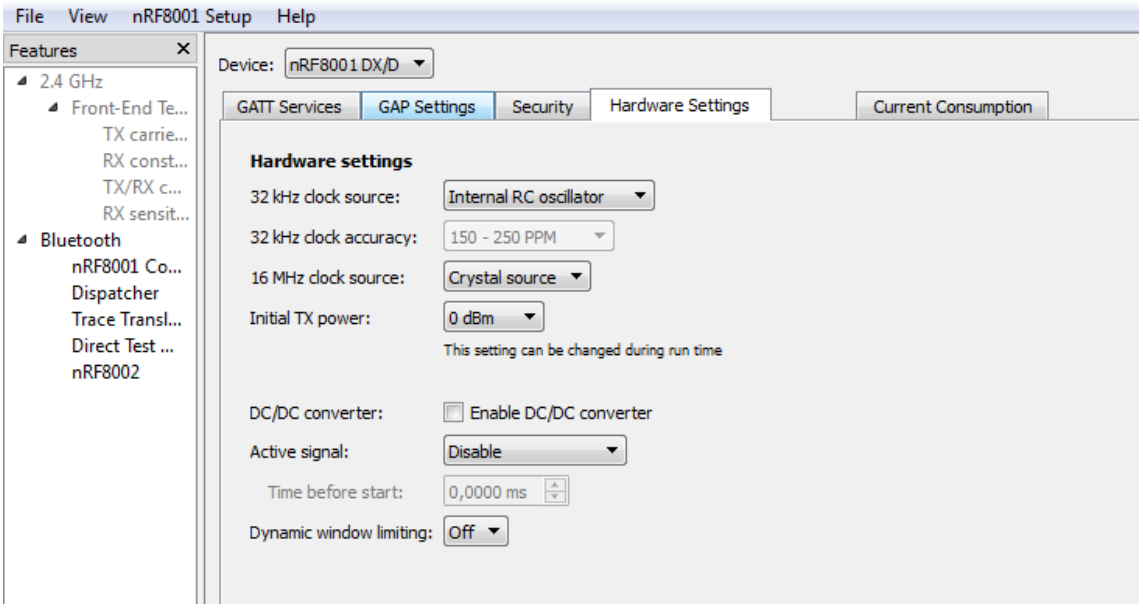


Figura 59. Pestaña Hardware Settings.

Pestaña Current Consumption

La pestaña Current Consumption permite estimar cual será el consumo de corriente del chip nRF8001 basándose en la configuración realizada, el tipo de uso del mismo y la capacidad de la batería utilizada. Esta pestaña es puramente informativa, habilitando campos modificables con el fin de ajustar más la estimación a los valores utilizados por la aplicación. Así, tanto en el intervalo que el nRF8001 espera entre conexiones sucesivas ("Connection interval") como el tiempo entre *advertising* (*Advertising interval*) serán establecidos por la aplicación del microcontrolador. En la parte inferior de la pestaña es posible ajustar el tiempo que el nRF8001 pasará conectado (estado Connected), dormido (modo Sleep) y Advertising (modo Advertising), con el fin de añadir precisión a la estimación realizada.

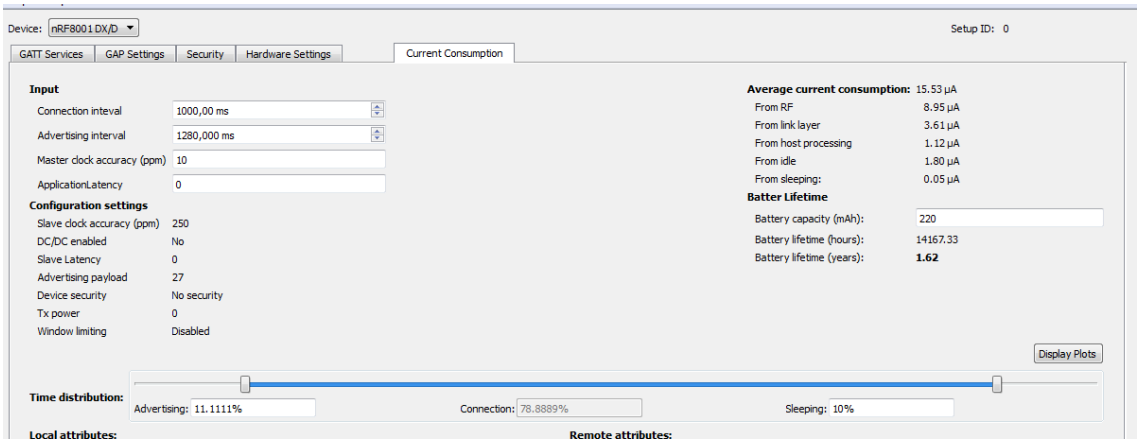


Figura 60. Pestaña Current Consumption.

Una vez finalizada la configuración del nRF8001 mediante el software nRFgoStudio, como se puede apreciar en la figura 61 el paso relativo a "Computer software tool nRFgo Studio" estará completado. El siguiente paso será "Import configuration setup and configure nRF8001". Es hora de generar los archivos que se utilizaran para cargar la configuración en el chip. Mediante nRF8001 Setup -> Generate Source files se generaran todos los archivos necesarios. Se generarán 3 archivos: services.h, services_lock.h y ublue_setup.gen.out.

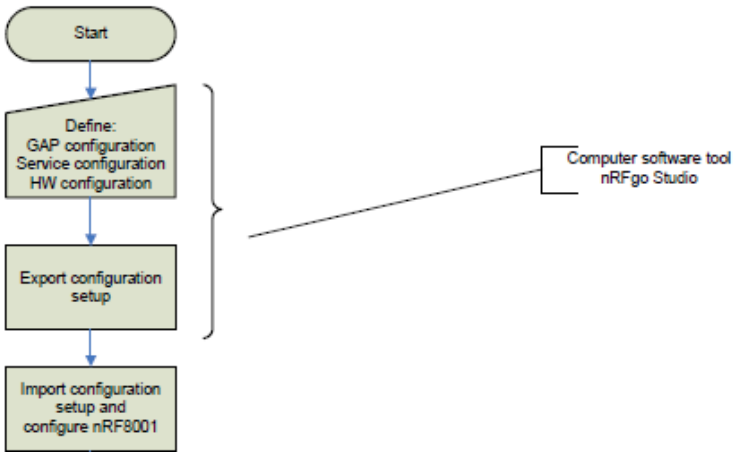


Figura 61. Diagrama de flujo del proceso de configuración del chip nRF8001. Fuente: Nordic Semiconductor

De estos tres archivos solo se utilizará service.h, el cual contiene una serie de mensajes generados por el programa a partir de la configuración realizada. Mediante el envío de esta batería de mensajes de tipo "Setup" al nRF8001 este quedará finalmente configurado. La diferencia entre services.h y services_lock.h es que si se envía la batería de mensajes incluidos en este último la configuración será cargada en la memoria no volátil del chip y no podrá volver a ser modificada. En el siguiente apartado se explicara cómo se produce este envío.

Importar la configuración en el nRF8001

El procedimiento de configuración del nRF8001 se realizara mediante el envío de los comandos de tipo "Setup" que han sido generados mediante nRFgo Studio y escritos en services.h. En la figura 62 se puede apreciar el flujo de mensajes requerido para importar la configuración en el nRF8001. Como se vio en la sección **Mecanismos para el control del flujo** en primer lugar un evento del tipo *DeviceStartedEvent* es recibido por parte del microcontrolador, el cual es enviado cuando el nRF8001 ha cambiado de modo de operación o ha sido reiniciado.

Después, el microcontrolador deberá empezar a enviar uno a uno todos los comandos "Setup" generados por el nRFgo Studio, los cuales serán respondidos por un *CommandResponseEvent* con código 0x01, lo cual equivale a ACI_STATUS_TRANSACTION_CONTINUE ([37] Página 158). Una vez enviando el último

comando el chip responderá con un *CommandResponseEvent* con código 0x02 (ACI_STATUS_TRANSACTION_COMPLETE) seguido de un *DeviceStartedEvent*, el cual indicara que el nRF8001 ha pasado del modo Setup al modo Active en estado Standby, finalizando así la configuración del chip nRF8001**.** En la figura 62 también se muestra las dos alternativas a la hora de importar la configuración al nRF8001:

1) Utiliza los comandos Setup de services.h, el nRF8001 guardará la configuración en la memoria volátil.
2) Utiliza los comandos Setup de services_lock.h, el nRF8001 guardará la configuración en la memoria no volátil.

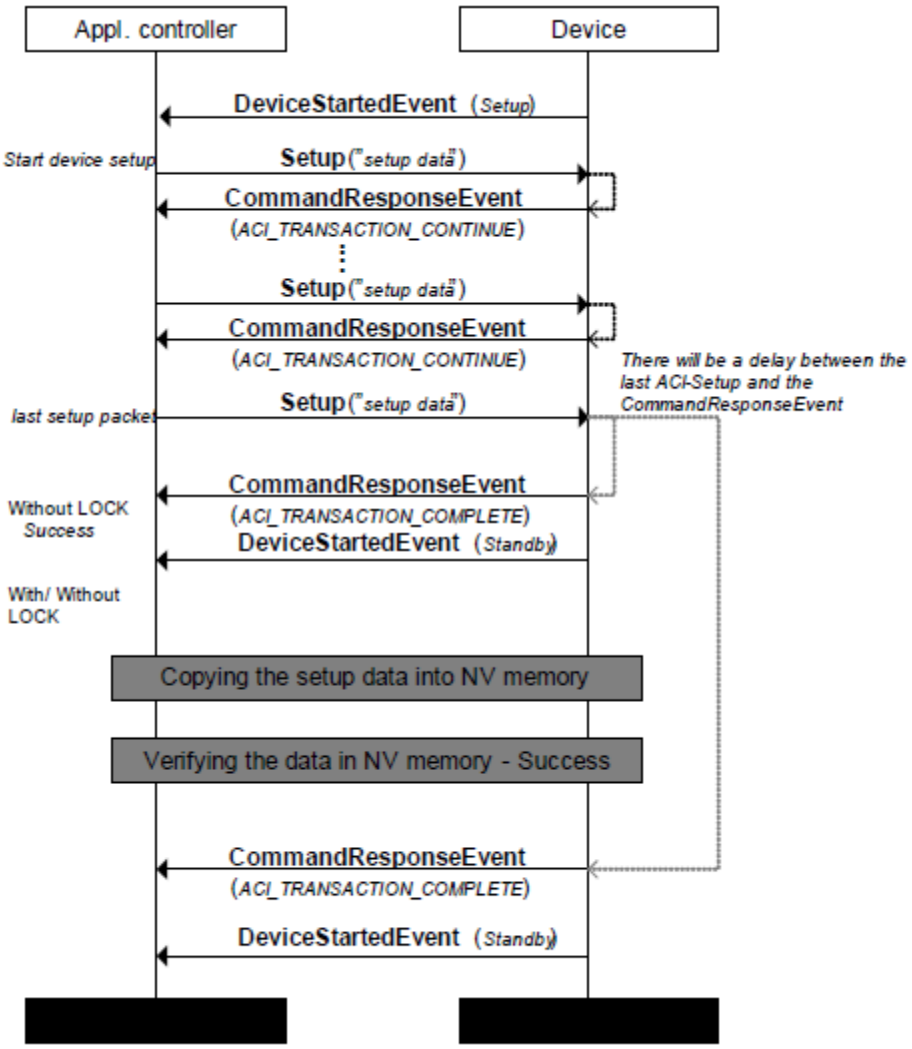


Figura 62. Fase final de la configuración en el nRF8001. Fuente: Nordic Semiconductor.

4.3. Comunicaciones entre el módulo Bluetooth nRF2740 y el dispositivo remoto

Siguiendo el diagrama de las figuras 61 y 63, una vez configurado el nRF8001 y establecida la conexión entre el chip y un *peer device* o dispositivo remoto, así como el *service discovery* cuando es necesario, la transferencia de datos entre el modulo y el dispositivo remoto se hará simplemente enviando comandos de tipo *SendData*.

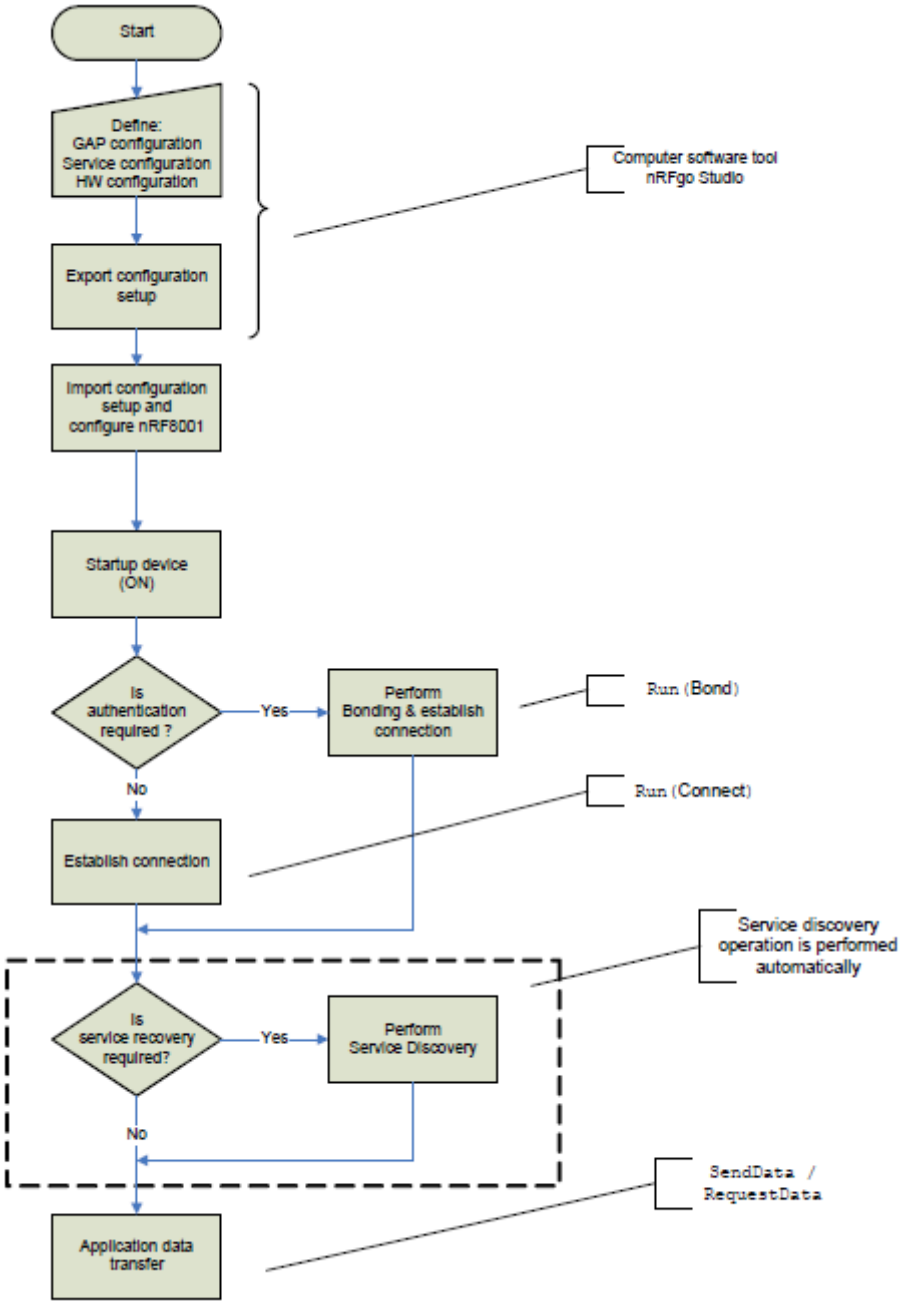


Figura 63. Procedimiento estándar de configuración y establecimiento de conexión. Fuente: Nordic Semiconductor

Estos comandos serán ejecutados a través de una *transmit pipe with ACK* como se vio en la figura 47. Una vez conocido como se ejecutan las comunicaciones es importante conocer el formato de los datos de la aplicación que se están enviando.

4.3.1. Estructura de los comandos SendData utilizados para enviar datos de la aplicación

Como se explicó en la sección **Estructura de los paquetes** cada paquete contiene una cabecera de 2 bytes en la que el primer byte contiene la longitud total del paquete (en bytes). El segundo byte contiene el código especifico que identifica al comando / evento enviado, en este caso el código identificativo del comando *SendData* (0x15).

En cuanto a la carga útil del paquete, los comandos *SendData* distribuyen la misma de la siguiente manera: 1 byte para almacenar el código identificativo de la *service pipe* que se va a utilizar para transmitir los datos, y hasta 20 bytes extra destinados al mensaje que se quiere transmitir.

4.3.2 Formato del mensaje

En lo referente a los 20 bytes destinados al mensaje, la especificación Bluetooth detalla cual será el contenido del mensaje en función de que lo que se quiera transmitir. Como se va a transmitir el valor de la característica Bluetooth *Temperature Measurement*, se accede a la especificación Bluetooth [51], la cual determina que:

La característica *Temperature Measurement* es una estructura de longitud variable que debe contener (en este orden) un *Flag Field*, un *Temperature Measurement Value field* y, en función del contenido del *Flag Field*, de manera opcional un *Time Stamp field* y/o un *Temperature Type field*.

El *Flag Field* será el primero en ser colocado en el mensaje y se compone de 2 bits, existiendo 4 combinaciones que pueden ser enviadas del mismo:

1) 00: Si ambos bits tienen valor 0 la estructura de la característica *Temperature Measurement* se compondrá de 2 campos, *Flags* y *Temperature Measurement Value* (en este orden).

2) 01: En este caso la estructura de la característica *Temperature Measurement* se compondrá de 3 campos: *Flags*, *Temperature Measurement Value* y *Temperature Type* (en este orden).

3) 10: La estructura de la característica *Temperature Measurement* se compondrá de 3 campos: *Flags*, *Temperature Measurement Value* y *Time Stamp* (en este orden).

4) 11: Si ambos bits tienen valor 1 la estructura de la característica *Temperature Measurement* se compondrá de 4 campos, *Flags*, *Temperature Measurement Value*, *Time Stamp* y *Temperature Type* (en este orden).

En este caso se utilizará un *Flag* de 00, enviando *Flags* y *Temperature Measurement Value*, ¿**por qué no se enviará un campo de *Time Stamp* si en el diseño de la aplicación se ha especificado que debe mostrarlo**?

Para poder enviar una marca de tiempo fiable desde el nRF8001 se necesitaría o bien un módulo extra RTC (Real Time Clock), un chip GPS o conexión a Internet. Todas las alternativas encarecerían el sistema y la conexión a internet además de encarecer el sistema teniendo que adquirir un módulo Wi-Fi para la placa de desarrollo o un módulo GPRS no permitiría que fuera portable en el caso de requerir Wi-Fi, teniendo que disponer de señal cada vez que se quisiese utilizar.

Por tanto, la opción más favorable es utilizar el dispositivo remoto, dejando a la aplicación Android que obtenga la marca de tiempo y la muestre junto al valor de la temperatura recibida. En este punto surge otra cuestión, **¿será válida la marca de tiempo al recibir la medición o el intervalo de tiempo entre el envío y la recepción obligarían a tener que ajustar la misma?** La respuesta es afirmativa, la marca de tiempo sería válida ya que el tiempo que precisa un dispositivo BLE para establecer una conexión y transferir la información es igual o inferior a 4ms [55].

Volviendo al formato del mensaje, el siguiente campo a transmitir será el *Temperature Measurement Value*, que según [51] es un campo de tipo Float. Para la especificación Bluetooth, los valores de tipo Float deben seguir un formato IEEE-11073 32-bit FLOAT. Por tanto habrá que adaptar el valor medido a ese formato. Para ello, se utilizará lo

especificado en [54]. Para explicar mejor como se "fabricara" el contenido del campo *Temperature Measurement Value*, se utilizará un ejemplo:

El campo se compone siempre de 4 bytes, 1 byte le corresponde al exponente y 3 a la mantisa (en este orden).

	Exponent	Mantissa
Size	1 octet	3 octets

Figura 64. Formato 32-BIT FLOATING POINT DATA TYPE (FLOAT-TYPE). Fuente [54]

Suponiendo que se quiera transmitir la temperatura medida 32.25 °C:

$$32.25\ x\ 100 = 3225$$

$$Exponente = 10^{-2} = -2 = 0xFEh$$

$$Mantisa = 3225 = 0xC99h$$

Por tanto el campo *Temperature Measurement Value* a transmitir será:

0xFE 00 0C 99

Otro ejemplo, suponiendo que se quiera transmitir la temperatura medida 57.0 °C:

$$57.0\ x\ 10 = 570$$

$$Exponente = 10^{-1} = -1 = 0xFFh$$

$$Mantisa = 570 = 0x23Ah$$

Por tanto el campo *Temperature Measurement Value* a transmitir será:

0xFF 00 02 3A

Conocido el contenido del *Flag Field* y del *Temperature Measurement Value Field*, el mensaje completo será (en el caso de enviar 57.0 °C):

0x00FF00023A

4.4 Aplicación software del sistema

La aplicación software del sistema se encargará de manejar las comunicaciones entre microcontrolador, sensor de temperatura, chip nRF8001 y dispositivo remoto.

A la hora de realizar la aplicación software del sistema conviene destacar que el fabricante del chip nRF8001, Nordic Semiconductor, pone a disposición de los desarrolladores un SDK (software development kit) gratuito especialmente desarrollado para aplicaciones BLE en Arduino [56].

El SDK contiene ejemplos de aplicaciones del nRF8001 (entre ellas una aplicación de *Health Thermometer*) así como librerías escritas en C fácilmente portables a cualquier microcontrolador. Estas librerías están desarrolladas expresamente para el nRF8001 y en constante crecimiento y mejora. Tanto estas librerías como el código desarrollado en el apartado "Implementación de las comunicaciones entre sensor de temperatura y microcontrolador" en el que se definió la aplicación software que controlaba el sensor TC74, serán utilizados en esta aplicación.

Como toda aplicación o "*sketch"* programado mediante Arduino IDE, la aplicación está dividida en dos partes principales, *Setup* y *Loop*. Todo el código ubicado dentro del *Setup* solo será ejecutado una vez al reiniciar el microcontrolador, mientras que el código dentro de *Loop* será ejecutado de manera continua como si de un bucle se tratase (figura 65).

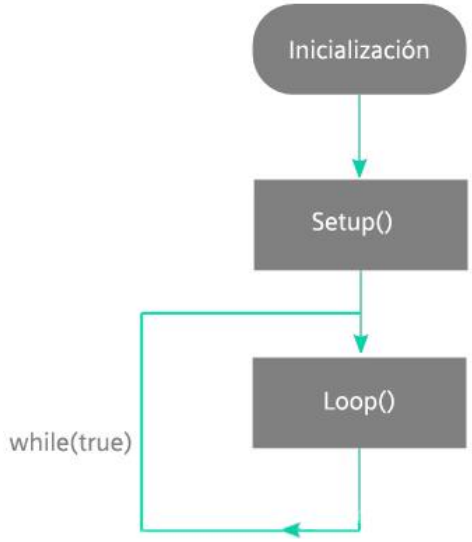


Figura 65. Diagrama de flujo de la aplicación software del sistema simplificado.

Inicialización

En la etapa de inicialización, la aplicación incluirá todas las librerías que se van a utilizar así como las variables del sistema. Se utilizarán dos librerías del SDK de Nordic, "lib_aci" y "aci_setup". La primera de ellas proporcionará a la aplicación métodos para implementar la interfaz ACI: envío de comandos, recepción de eventos, mecanismos para el control del flujo, etc. La segunda contiene métodos para realizar el proceso de importación de la configuración del nRF8001 a partir del archivo "services.h" generado. En cuanto al código programado para la aplicación software que controla las comunicaciones con el sensor de temperatura, este también será utilizado como librería de la aplicación principal.

Setup

En esta etapa se configurarán todos los detalles relativos al bus de comunicaciones SPI utilizado para comunicar con el chip nRF8001: pines, modo de funcionamiento del bus, frecuencia de reloj, etc.

Loop

Aquí se encuentra la práctica totalidad de código que define el comportamiento del sistema. El diagrama de flujo (figura 66) muestra de manera simplificada el funcionamiento de la aplicación, pero existen varios aspectos que es preciso detallar:

• El chip nRF8001 realizará *advertising* enviando paquetes cada 100ms durante 15 segundos. Una vez finalizado pasará a modo Sleep junto al microcontrolador, en el cual estará 30 segundos.

• Con el fin de asegurar que el valor medido de temperatura se ajusta en lo posible al valor real y evitar picos no deseados, la operación "Leer temperatura" toma 8 muestras y realiza la media. El resultado es el enviado al dispositivo remoto.

• El tiempo programado entre cada envío de valores de Tª es de 10 segundos.

• Antes de realizar cualquier tipo de envío, la aplicación comprueba si existen créditos disponibles y si la *service pipe* a utilizar se encuentra libre (ver apartado **Mecanismos para el control del flujo**).

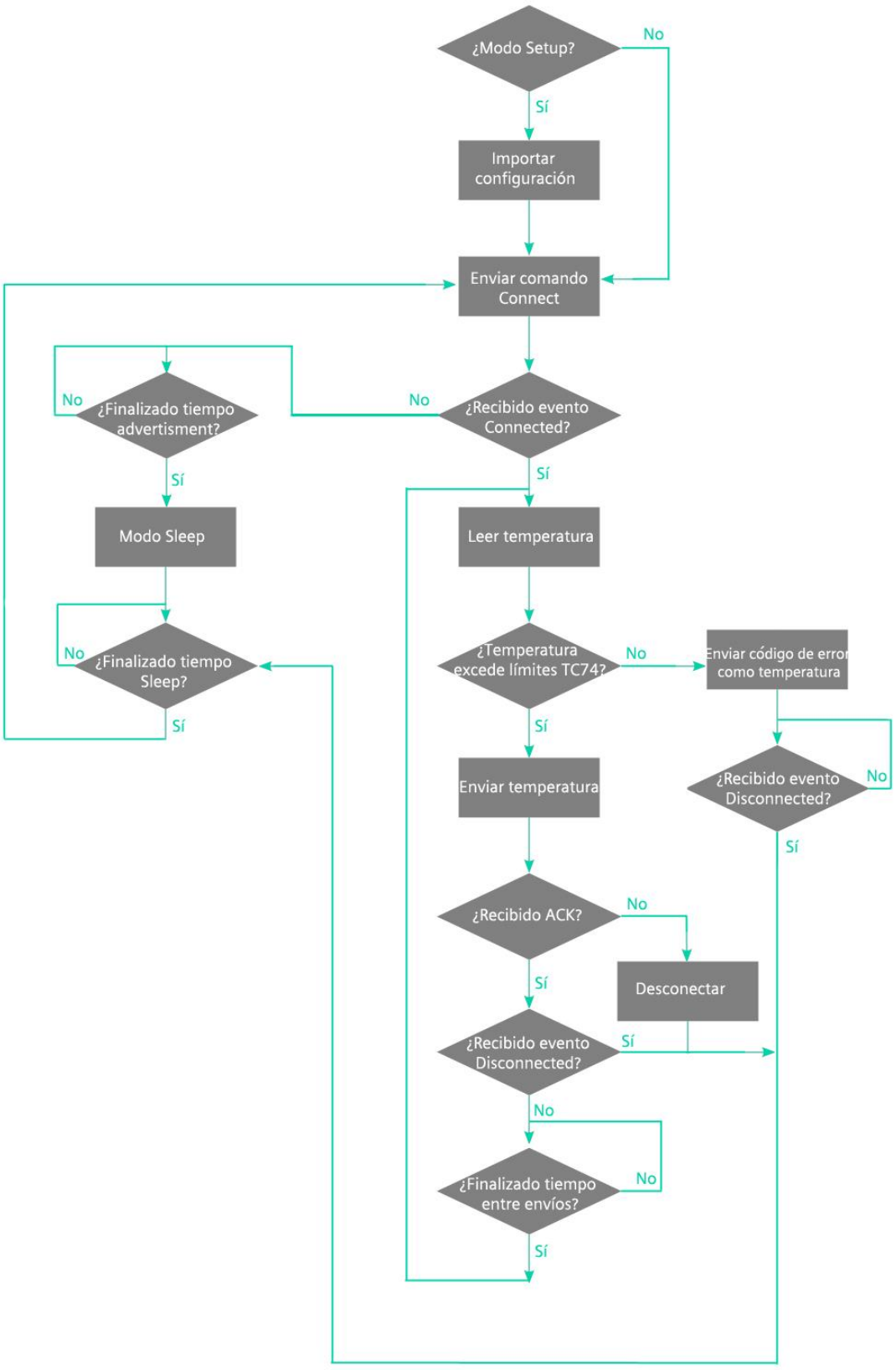


Figura 66. Diagrama de flujo de Loop.

4.5 Prototipo del sistema y solución final

En este apartado se muestra el aspecto del prototipo del sistema utilizado así como su posterior montaje en la versión más compacta y portable que conforma la solución final. En la figura 67 se pueden apreciar los diferentes componentes del prototipo del sistema, desde el sensor de temperatura y el conversor de niveles bidireccional de 4 canales ubicados en primera instancia, al módulo Bluetooth (verde) y la placa de desarrollo Arduino Micro (azul) ubicados en la parte superior de la imagen.

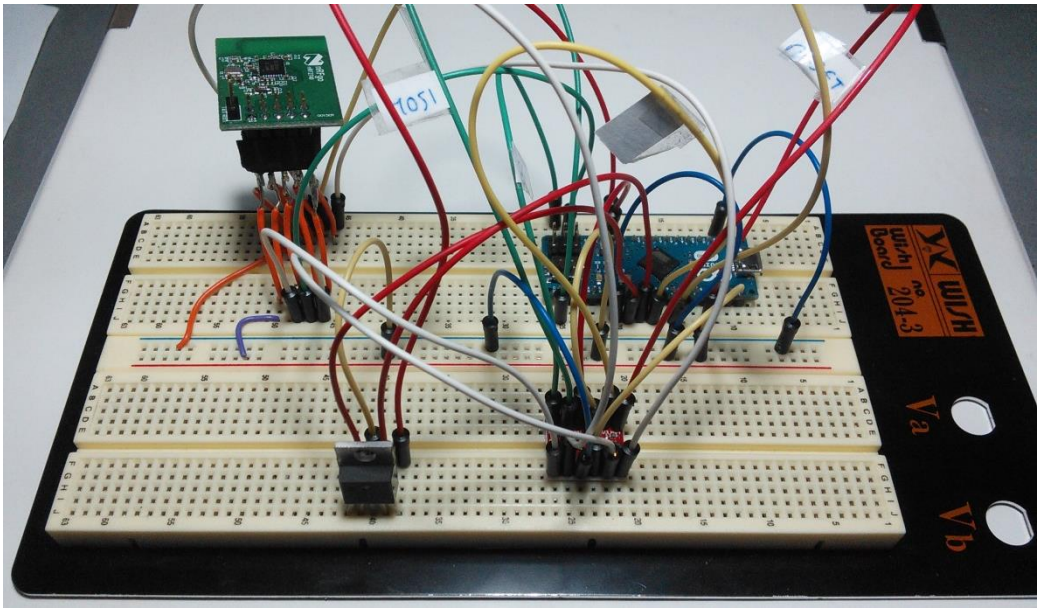


Figura 67. Prototipo del sistema.

A continuación se muestra el aspecto de la solución final del sistema, la cual incluye una carcasa modificada para albergar el sensor de temperatura y espacio para una batería de 9V.

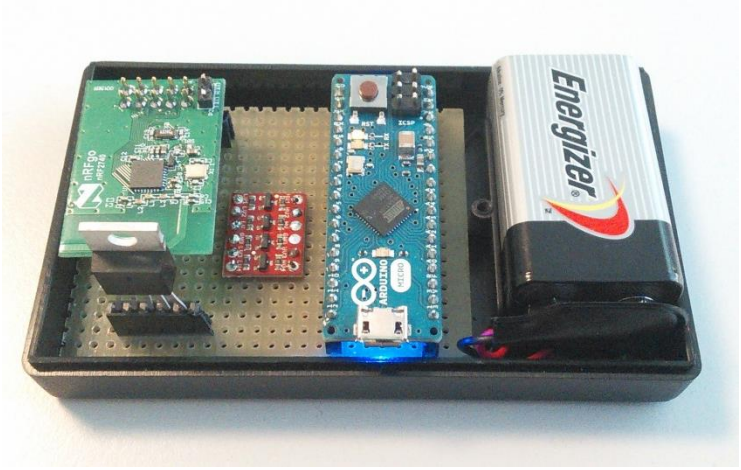


Figura 68. Aspecto final del sistema (1).

Figura 69. Aspecto final del sistema (2).

Figura 70. Aspecto final del sistema (3).

4.6 Desarrollo e implementación de la aplicación Android

En el apartado 3.4.3. se analizaron las alternativas en cuanto a sistemas operativos y las funcionalidades que debería tener la aplicación para dispositivos móviles del sistema. En este apartado se explicará paso a paso como se ha desarrollado la aplicación Android para cumplir esas funcionalidades o requerimientos. En primer lugar se analizará la arquitectura del sistema operativo elegido y las herramientas software que se utilizaran para desarrollar la aplicación.

4.6.1 Arquitectura de Android

El siguiente gráfico muestra la arquitectura de Android. Como se puede ver está formada por cuatro capas. Una de las características más importantes es que todas las capas están basadas en software libre.

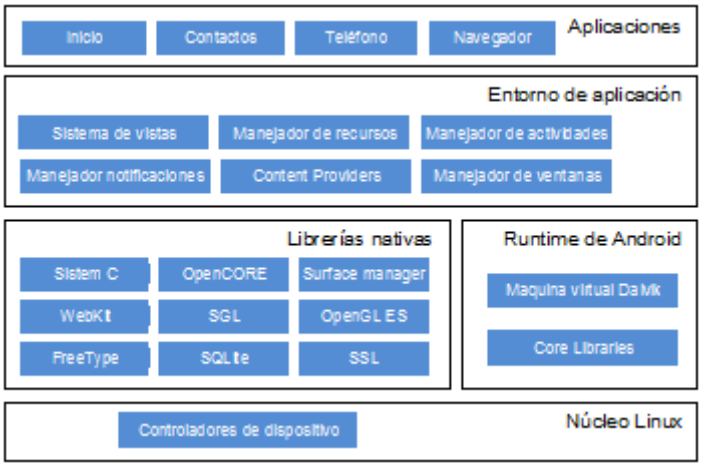


Figura 71. Arquitectura de Android. Fuente [21]

El núcleo Linux

El núcleo de Android está formado por el sistema operativo Linux versión 2.6. Esta capa proporciona servicios como la seguridad, el manejo de la memoria, el multiproceso, la pila de protocolos y el soporte de *drivers* para dispositivos.

Esta capa del modelo actúa como capa de abstracción entre el hardware y el resto de la pila. Por lo tanto, es la única que es dependiente del *hardware*.

***Runtime* de Android**

Está basado en el concepto de máquina virtual utilizado en Java. Dado las limitaciones de los dispositivos donde ha de correr Android (poca memoria y procesador limitado) no fue posible utilizar una máquina virtual Java estándar. Google tomó la decisión de crear una nueva, la máquina virtual Dalvik, que respondiera mejor a estas limitaciones.

Algunas características de la máquina virtual Dalvik que facilitan esta optimización de recursos son: ejecuta ficheros Dalvik ejecutables (.dex) –formato optimizado para ahorrar memoria. Además, está basada en registros. Cada aplicación corre en su propio proceso Linux con su propia instancia de la máquina virtual Dalvik. Delega al kernel de Linux algunas funciones como *threading* y el manejo de la memoria a bajo nivel.

También se incluye en el *Runtime de Android* las "core libraries" con la mayoría de las librerías disponibles en el lenguaje Java.

Librerías nativas

Incluye un conjunto de librerías en C/C++ usadas en varios componentes de Android. Están compiladas en código nativo del procesador. Muchas de las librerías utilizan proyectos de código abierto. Algunas de estas librerías son:

- **System C library**: una derivación de la librería BSD de C estándar (libc), adaptada para dispositivos embebidos basados en Linux.
- **Media Framework**: librería basada en PacketVideo's OpenCORE; soporta codecs de reproducción y grabación de multitud de formatos de audio vídeo e imágenes MPEG4, H.264, MP3, AAC, AMR, JPG y PNG.
- **Surface Manager**: maneja el acceso al subsistema de representación gráfica en 2D y 3D.
- **WebKit**: soporta un moderno navegador Web utilizado en el navegador Android y en la vista Webview. Se trata de la misma librería que utiliza Google Chrome y Safari de Apple.
- **SGL**: motor de gráficos 2D.

- **Librerías 3D**: implementación basada en OpenGL ES 1.0 API. Las librerías utilizan el acelerador hardware 3D si está disponible, o el software altamente optimizado de proyección 3D.
- **FreeType**: fuentes en bitmap y renderizado vectorial.
- **SQLite**: potente y ligero motor de bases de datos relacionales disponible para todas las aplicaciones.
- **SSL**: proporciona servicios de encriptación *Secure Socket Layer*.

Entorno de aplicación

Proporciona una plataforma de desarrollo libre para aplicaciones con gran riqueza e innovaciones (sensores, localización, servicios, barra de notificaciones,).

Esta capa ha sido diseñada para simplificar la reutilización de componentes. Las aplicaciones pueden publicar sus capacidades y otras pueden hacer uso de ellas (sujetas a las restricciones de seguridad). Este mismo mecanismo permite a los usuarios reemplazar componentes.

Una de las mayores fortalezas del entorno de aplicación de Android es que se aprovecha el lenguaje de programación Java. El SDK de Android no acaba de ofrecer todo lo disponible para su estándar del entorno de ejecución Java (JRE), pero es compatible con una fracción muy significativa de la misma.

Los servicios más importantes que incluye son:

- **Views**: extenso conjunto de vistas, (parte visual de los componentes).
- **Resource Manager**: proporciona acceso a recursos que no son en código.
- **Activity Manager**: maneja el ciclo de vida de las aplicaciones y proporciona un sistema de navegación entre ellas.
- **Notification Manager**: permite a las aplicaciones mostrar alertas personalizadas en la barra de estado.

- **Content Providers**: mecanismo sencillo para acceder a datos de otras aplicaciones (como los contactos).

Aplicaciones

Este nivel está formado por el conjunto de aplicaciones instaladas en una máquina Android y será el nivel donde se ejecute la aplicación diseñada. Todas las aplicaciones han de correr en la máquina virtual Dalvik para garantizar la seguridad del sistema.

4.6.2 Herramientas software utilizadas para desarrollar la aplicación

Normalmente las aplicaciones Android están escritas en Java. Para desarrollar la aplicación del sistema se utilizará el paquete que Google pone a disposición de los desarrolladores llamado *Eclipse ADT bundle*, el cual incluye:

- Eclipse + ADT plugin.
- Android SDK Tools.
- Android Platform-tools.
- Una versión de la plataforma Android.
- Emulador de dispositivos Android.

Del cual los componentes principales son:

Android SDK

Ofrecido gratuitamente por Google, ofrece una serie de drivers, herramientas y recursos diversos para programar en Android. En Android SDK se incluyen las herramientas necesarias para empezar a programar para esta plataforma: distintas APIs facilitadas por Google tanto para el control de las funciones del dispositivo como para la integración de servicios, un depurador, un emulador para testear las aplicaciones y multitud de documentación que cubre la gran mayoría de las funcionalidades de este sistema operativo.

ADT Plugin

El plugin ADT (Android Development Tools) es un plugin para el IDE Eclipse específicamente diseñado para ofrecer un entorno donde programar aplicaciones para Android.

ADT extiende las capacidades de Eclipse adaptándolas al sistema Android, permitiendo al usuario crear de manera rápida y sencilla nuevos proyectos para Android, añadir librerías y ejemplos basados en el sistema operativo, depurar los proyectos utilizando Android SDK Tools e incluso exportar archivos .apk con el fin de distribuir las aplicaciones creadas.

Eclipse

Eclipse es un programa informático compuesto por un conjunto de herramientas de programación de código abierto multiplataforma desarrollado originalmente por IBM.

El entorno de desarrollo integrado (IDE) de Eclipse emplea módulos (en inglés *plug-in*) para proporcionar diferentes funcionalidades en base a los requerimientos del usuario, a diferencia de otros entornos donde las funcionalidades están todas incluidas, las necesite el usuario o no. Este mecanismo de módulos es una plataforma ligera para componentes de software. Se provee soporte para Java y CVS en el SDK de Eclipse. Y no tiene por qué ser usado únicamente con estos lenguajes, ya que soporta otros lenguajes de programación.

El SDK de Eclipse incluye las herramientas de desarrollo de Java, ofreciendo un IDE con un compilador de Java interno y un modelo completo de los archivos fuente de Java. Esto permite técnicas avanzadas de refactorización y análisis de código. Mediante diversos plugins estas herramientas están también disponibles para otros lenguajes como C/C++ y en la medida de lo posible para lenguajes de script como PHP o Javascript.

4.6.3 Ficheros y carpetas de un proyecto Android

Un proyecto Android está formado básicamente por un descriptor de la aplicación (*AndroidManifest.xml*), el código fuente en Java y una serie de ficheros con recursos.

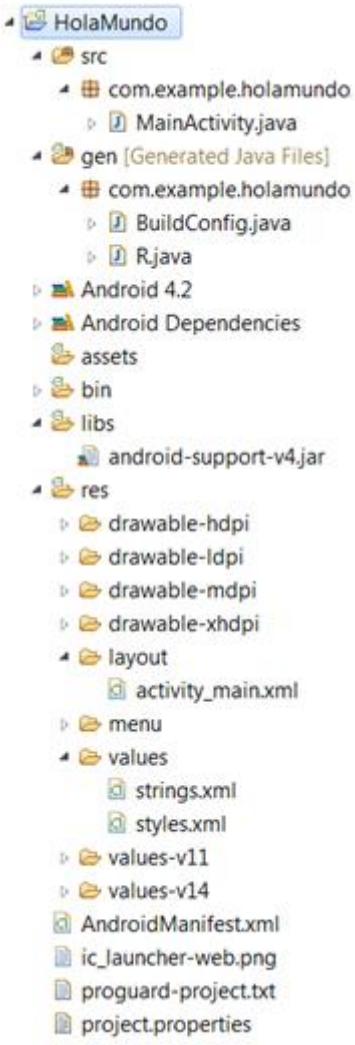


Figura 72. Vista de las carpetas de las que se compone un proyecto Android. Fuente [21].

Cada elemento se almacena en una carpeta específica:

src: Carpeta que contiene el código fuente de la aplicación. Como se puede observar los ficheros Java se almacenan en un espacio de nombres.

gen: Carpeta que contiene el código generado de forma automática por el SDK. Nunca hay que modificar de forma manual estos ficheros. Dentro se encuentra:

-BuildConfig.java: Define la constante DEBUG para que desde Java se pueda saber si la aplicación está en fase de desarrollo.

-R.java: Define una clase que asocia los recursos de la aplicación con identificadores. De esta forma los recursos podrán ser accedidos desde Java.

Android x.x: Código JAR, el API de Android según la versión seleccionada.

Android Private Libraries: Librerías asociadas al proyecto.

assets: Carpeta que puede contener una serie arbitraria de ficheros o carpetas que podrán ser utilizados por la aplicación (ficheros de datos, fuentes,…). A diferencia de la carpeta res, nunca se modifica el contenido de los ficheros de esta carpeta ni se les asociará un identificador.

bin: En esta carpeta se compila el código y se genera el .apk, fichero comprimido que contiene la aplicación final lista para instalar.

libs: Código JAR con librerías que se usaran en el proyecto. Se añade automáticamente la librería android-support-v4. Su objetivo es permitir ciertas funcionalidades importantes no disponibles en el nivel de API seleccionado como mínimo. Gracias a esta librería se puede utilizar elementos como *Fragments, ViewPager* o *Navigation Drawable*, que no están disponibles en el nivel de API mínimo seleccionado.

res: Carpeta que contiene los recursos usados por la aplicación. Las subcarpetas pueden tener un sufijo si se quiere que el recurso solo se cargue al cumplirse una condición. Por ejemplo –hdpi significa que solo ha de cargar los recursos contenidos en esta carpeta cuando el dispositivo donde se instala la aplicación tiene una densidad gráfica alta (>180dpi); -v11significa que el recurso solo ha de cargarse en un dispositivo con nivel de API 11 (v3.0).

-drawable: En esta carpeta se almacenan los ficheros de imágenes (JPG o PNG) y descriptores de imágenes en XML.

-layout: Contiene ficheros XML con vistas de la aplicación. Las vistas permiten configurar las diferentes pantallas que compondrán la interfaz de

usuario de la aplicación. Se utiliza un formato similar al HTML usado para diseñar páginas Web. Serán tratadas en el siguiente capítulo.

-menu: Ficheros XML con los menús de cada actividad.

-values: También se utilizaran ficheros XML para indicar valores de tipo string, color o estilo. De esta manera se pueden cambiar los valores sin necesidad de ir al código fuente. Por ejemplo, permitiría traducir una aplicación a otro idioma.

-anim: Contiene ficheros XML con animaciones.

-animator: Contiene ficheros XML con animaciones de propiedades.

-xml: Otros ficheros XML requeridos por la aplicación.

-raw: Ficheros adicionales que no se encuentran en formato XML.

AndroidManifest.xml: Este fichero describe la aplicación Android. En él se indican las actividades, intenciones, servicios y proveedores de contenido de la aplicación. También se declaran los permisos que requerirá la aplicación. Se indica la versión mínima de Android para poder ejecutarla, el paquete Java, la versión de la aplicación, etc.

ic_launcher-web.png: Icono de la aplicación de gran tamaño para ser usado en páginas Web. El nombre puede variar si se indicó uno diferente en el proceso de creación del proyecto.

proguard-project.txt: Fichero de configuración de la herramienta ProGuard, que te permite optimizar y ofuscar el código generado. Es decir, se obtiene un .apk más pequeño y donde resulta más difícil hacer ingeniería inversa.

project.properties: Fichero generado automáticamente por el SDK. Nunca hay que modificarlo. Se utiliza para comprobar la versión del API y otras características cuando se instala la aplicación en el terminal.

4.6.4. Versión de la aplicación

Una vez comprendida la arquitectura del sistema operativo Android y las herramientas que se van a utilizar para el desarrollo de la aplicación el siguiente paso es decidir cuál será la versión mínima de Android que soporte la aplicación, es decir, los dispositivos que tengan instalada una versión de Android inferior a la establecida como mínima, no podrán instalar la aplicación ni verla en la Google Play Store en el caso de que esta esté subida a la misma.

Como se explicó en apartados anteriores, existe una restricción relativa a la versión Android de la aplicación, y es que esta no puede ser inferior a la versión 4.3 (API level 18) ya que esta versión ha sido la que ha introducido un soporte de plataforma ya integrado para BLE. Por tanto la elección es clara, estableciendo que la versión mínima que soportara la aplicación será la 4.3. En cuanto a la versión máxima, a día de hoy solo existen dos opciones, 4.3 o 4.4. A primera vista lo lógico sería optar por la versión más avanzada para intentar que la aplicación sea compatible con el máximo número de dispositivos. Sin embargo, al ser una aplicación desarrollada en exclusiva para un sistema específico, se elegirá la versión más estable o que ofrezca mejores prestaciones para nuestro sistema. Es por ello que se deberá analizar primero que ofrece cada versión y elegir entre ellas.

Android 4.3 Nivel de API 18 (julio 2013)

Esta versión introduce mejoras en múltiples áreas. Entre ellas los perfiles restringidos (disponible sólo en tabletas) que permiten controlar los derechos de los usuarios para ejecutar aplicaciones específicas y para tener acceso a datos específicos. Igualmente, los programadores pueden definir restricciones en las apps, que los propietarios puedan activar si quieren. Se da soporte para Bluetooth Low Energy (BLE) que permite a los dispositivos Android comunicarse con los periféricos con bajo consumo de energía. Se agregan nuevas características para la codificación, transmisión y multiplexación de datos multimedia. Se da soporte para OpenGL ES 3.0. Se mejora la seguridad para gestionar y ocultar las claves privadas y credenciales.

Android 4.4 Nivel de API 19 (octubre 2013)

Aunque se esperaba la versión número 5.0 y con el nombre *Key Lime Pie*, Google sorprendió con el cambio de nombre, que se debió a un acuerdo con Nestlé para asociar ambas marcas.

Un objetivo principal de la versión 4.4 es hacer Android disponible en una gama aún más amplia de dispositivos, incluyendo aquellos con tamaños de memoria RAM de sólo 512 MB . Para ello, todos los componentes principales de Android han sido recortados para reducir sus requerimientos de memoria, y se ha creado una nueva API que permite adaptar el comportamiento de la aplicación en dispositivos con poca memoria.

Más visibles son algunas nuevas características de la interfaz de usuario. El modo de inmersión en pantalla completa oculta todas las interfaces del sistema (barras de navegación y de estado) de tal manera que una aplicación puede aprovechar el tamaño de la pantalla completa. WebViews (componentes de la interfaz de usuario para mostrar las páginas Web) se basa ahora en el software de Crome de Google y por lo tanto puede mostrar contenido basado en HTM5.

Se mejora la conectividad con soporte de NFC para emular tarjetas de pago tipo HCE, varios protocolos sobre Bluetooth y soporte para mandos infrarrojos. También se mejoran los sensores para disminuir su consumo y se incorpora un sensor contador de pasos.

Se facilita el acceso de las aplicaciones a la nube con un nuevo marco de almacenamiento. Este marco incorpora un tipo específico de *content provider* conocido como *document provider*, nuevas intenciones para abrir y crear documentos y una ventana de dialogo que permite al usuario seleccionar ficheros. Se incorpora un administrador de impresión para enviar documentos a través de Wi-Fi a una impresora. También se añade un *content provider* para gestionar los SMS.

Desde una perspectiva técnica, hay que destacar la introducción la nueva máquina virtual ART, que consigue tiempos de ejecución muy superiores a la máquina Dalvik. Sin embargo, todavía está en una etapa experimental. Por defecto se utiliza la máquina

virtual de Dalvik, permitiendo a los programadores activar opcionalmente ART para verificar que sus aplicaciones funcionan correctamente.

Como se ve en la versión 4.4 se han añadido funcionalidades interesantes que pueden ser de utilidad para el sistema en futuras ampliaciones y mejoras, como la nueva máquina virtual ART. Además, la versión 4.4 es compatible con la 4.3, es decir, solo se añaden nuevas funcionalidades y en el caso de modificar alguna esta no se elimina, etiquetándose como obsoletas pero pudiéndose continuar utilizando, por lo que la versión máxima que soporte la aplicación será la 4.4, que por otro lado es la última versión disponible.

4.6.5 Implementación de la aplicación

Como se ha visto en el apartado anterior, en un proyecto Android cada elemento se almacena en una carpeta específica. A continuación se analizaran los elementos más relevantes del proyecto, llamado BLETD (Bluetooth Low Energy Temperature Display), empezando por la carpeta src, la cual contiene el código fuente de la aplicación.

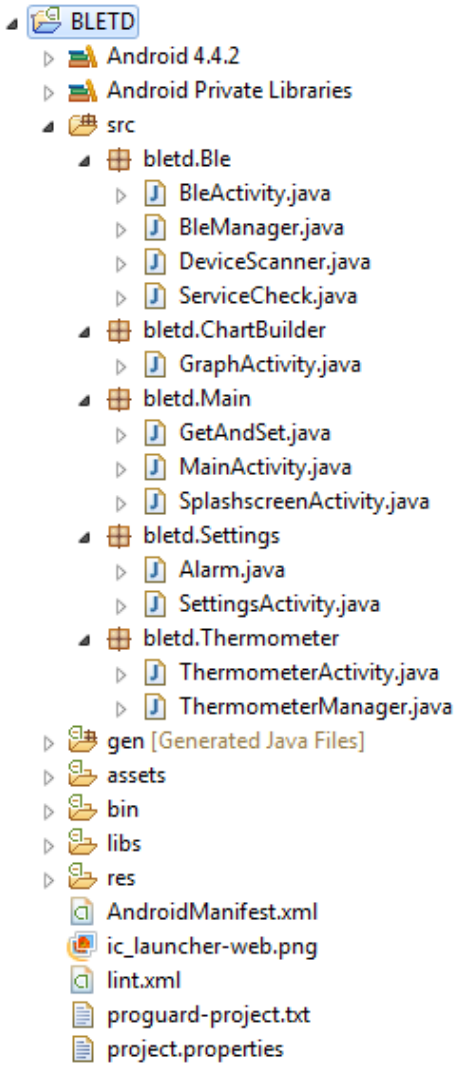


Figura 73. Vista de las carpetas de las que se compone la aplicación BLETD.

Dentro de la carpeta src se han creado varios paquetes java, cada uno de los cuales contiene las clases que se utilizarán en la aplicación. Estos paquetes permiten agrupar las distintas partes del programa cuya funcionalidad tienen elementos comunes.

bletd.Ble

En este paquete se incluyen todos los métodos y clases que se utilizarán para manejar las comunicaciones vía Bluetooth low energy. A continuación se explica el contenido de cada clase.

BleActivity

La clase BleActivity implementa las siguientes funciones:

- Comprobación inicial que asegura que el dispositivo en el cual se ha instalado la aplicación soporta Bluetooth low energy.
- Comprobación de si está o no activado el Bluetooth en caso de que el dispositivo soporte BLE.
- Mostrar en pantalla diferentes valores dinámicos incluidos dentro de la GUI (*graphical user interface*) utilizada por la aplicación: Nombre del dispositivo y botón CONECTAR / DESCONECTAR.
- Cambiar el estado del botón CONECTAR / DESCONECTAR, si el dispositivo se encuentra desconectado mostrar CONECTAR, en caso contrario, mostrar DESCONECTAR.
- Guardar el estado del botón CONECTAR / DESCONECTAR así como el nombre del dispositivo mostrado en la GUI.
- Comprobar si el Bluetooth se encuentra activado cuando el usuario pulsa el botón CONECTAR, si no se encuentra activado, mostrar ventana que solicita activación.
- Tratar como desconexión la perdida de comunicaciones con el nRF8001 o un error en las mismas, cambiando estado del botón CONECTAR / DESCONECTAR.
- Mostrar dialogo cuando el dispositivo no soporta Bluetooth low energy.
- Obtener el nombre del dispositivo Bluetooth al que se conecta la aplicación.

BleManager

La clase BleManager implementa las siguientes funciones:

- Iniciar el manager Bluetooth, el cual se utiliza para gestionar el acceso a servicios Bluetooth incluidos en el dispositivo conectado con la aplicación.
- Finalizar (destruir) el manager Bluetooth.
- Conectar con el dispositivo Bluetooth cuando el usuario pulsa el botón CONECTAR.
- Desconectar del dispositivo Bluetooth cuando el usuario pulsa el botón DESCONECTAR.
- Devolver un código de error cuando se ha producido algún error en la comunicación BLE, finalizando posteriormente las comunicaciones.
- Obtener la dirección del dispositivo con el que se ha realizado la conexión (en este caso del nRF8001).
- Obtener el nombre del dispositivo con el que se ha realizado la conexión (en este caso del nRF8001).
- Devolver un valor de tipo booleano *true* si se ha establecido una conexión entre el dispositivo y el nRF8001.

DeviceScanner

La clase DeviceScanner implementa las siguientes funciones:

- Inicia la búsqueda de dispositivos BLE (*scan*).
- Detener la búsqueda de dispositivos BLE.
- Analizar el paquete de *advertising* recibido: Comprobar si es posible conectar al dispositivo encontrado.
- Guardar el nombre, dirección y valor RSSI del dispositivo conectado.

- Mostrar el nombre, dirección y valor RSSI del dispositivo encontrado.
- Actualizar el valor de potencia de la señal Bluetooth recibida (RSSI).
- Mostrar el valor RSSI en la GUI en una imagen que varía dependiendo del nivel recibido.
- Añadir o actualizar el dispositivo Bluetooth encontrado: Si el dispositivo no existe en la lista de dispositivos Bluetooth conectados (en este caso solo contendrá un dispositivo, el nRF8001), añade el dispositivo a la lista. En caso contrario, actualiza el valor RSSI.

ServiceCheck

La clase ServiceCheck implementa las siguientes funciones:

- Analizar el paquete de *advertising* recibido: Comprobar si el dispositivo encontrado dispone de los UUID requeridos por la aplicación.
- Iniciar el procedimiento de *Service Discovery* una vez se han conectado el dispositivo y el chip nRF8001.
- Devolver un valor de tipo booleano *true* cuando el procedimiento *Service Discovery* ha finalizado y ha encontrado los servicios Bluetooth que utiliza la aplicación en el nRF8001, en caso contrario, devolver *false.*

Bletd.Thermometer

En este paquete se incluyen todos los métodos y clases que se utilizarán para manejar los datos recibidos relativos al servicio Bluetooth *Health Thermometer*. A continuación se explica el contenido de cada clase.

ThermometerManager

La clase ThermometerManager implementa las siguientes funciones:

- Leer el valor de la característica *Temperature Measurement*. Si el valor es igual al especificado como “Código de error enviado como temperatura”, el cual

indica que no se ha podido obtener una medición valida, enviar el código de error asociado. En caso contrario, almacenar el valor recibido.

- Notificar al dispositivo que el valor de la característica *Temperature Measurement* ha variado.
- Habilitar el envío del ACK cuando se recibe un valor valido de la característica *Temperature Measurement.*
- Enviar ACK.
- Dar formato al valor recibido de la característica *Temperature Measurement*.

ThermometerActivity

La clase Thermometer Activity es una de las tres actividades principales de la aplicación. Está asociada a la pestaña “Termómetro” e implementa las siguientes funciones:

- Mostrar en pantalla diferentes valores dinámicos incluidos dentro de la GUI utilizada por la aplicación: Valor de temperatura recibido y marca de tiempo.
- Actualizar el valor de temperatura y la marca de tiempo al recibir un nuevo valor de temperatura.
- Definir la interfaz por defecto cuando no existen valores de temperatura disponibles.
- Definir la marca de tiempo.
- Guardar en un archivo txt llamado “Log” almacenado en la memoria del dispositivo un registro con las temperaturas recibidas y su marca de tiempo asociada.

Bletd.ChartBuilder

En este paquete se incluyen todos los métodos y clases que se utilizarán para mostrar de manera gráfica en un diagrama los valores de temperatura recibidos y las marcas de tiempo asociadas a los mismos. A continuación se explica el contenido de cada clase.

GraphActivity

La clase GraphActivity es una de las tres actividades principales de la aplicación junto a ThermometerActivity y SettingsActivity. Está asociada a la pestaña "Gráfica" e implementa las siguientes funciones:

- Configuración del grafico mostrado. Permite cambiar todos los detalles gráficos: Colores, espaciado, rótulos, tipo de gráfico, zoom, etc.
- Asociar un identificador único a cada medida recibida.
- Actualizar el gráfico con el valor de temperatura recibido y la marca de tiempo asociada siempre y cuando no se pinte dos veces la información relativa al mismo identificador.

Bletd.Settings

En este paquete se incluyen todos los métodos y clases que se utilizarán configurar los ajustes de la aplicación definidos por el usuario. A continuación se explica el contenido de cada clase.

Alarm

La clase Alarm implementa las siguientes funciones:

- Configuración de la alarma sonora de la aplicación: Volumen, duración, repeticiones, nombre del archivo utilizado, etc.
- Reproducción en bucle del archivo de sonido asociado a la alarma sonora.

SettingsActivity

La clase SettingsActivity es una de las tres actividades principales de la aplicación junto a ThermometerActivity y GraphActivity. Está asociada a la pestaña “Ajustes” e implementa las siguientes funciones:

- Cargar en pantalla los ajustes guardados por el usuario de forma automática.
- Guardar en memoria los ajustes realizados para que estos estén disponibles en arranques sucesivos.
- Comprobar si el Wi-Fi se encuentra activo cuando se ha activado la notificación mediante correo electrónico. En caso de que no se encuentre activado, activar Wi-Fi.
- Devolver un valor de tipo booleano *true* si se ha activado la notificación mediante correo electrónico y el Wi-Fi se encuentra activo.
- Devolver un valor de tipo booleano *true* si se ha activado la notificación mediante SMS.
- Devolver un valor de tipo booleano *true* si se ha activado la notificación mediante alarma sonora.
- Enviar un correo electrónico a la dirección de destino desde la dirección especificada por el usuario.
- Enviar un SMS al número de teléfono especificado por el usuario.
- Comprobar cada vez que se recibe un valor de temperatura que no supera el valor límite superior o es inferior al valor límite inferior establecido por el usuario.

Bletd.Main

En este paquete se incluyen todos los métodos y clases que se utilizarán para controlar la transición entre pestañas y actividades de la aplicación. A continuación se explica el contenido de cada clase.

SplashScreenActivity

La clase SplashScreenActivity es la encargada de mostrar la pantalla de inicio y, al cabo de un tiempo definido, pasar a la actividad MainActivity.

MainActivity

La clase MainActivity define cual es el orden de las pestañas mostradas en la aplicación: “Ajustes”, “Termómetro” y “Gráfica”. Además implementa el mecanismo de transición entre ellas y las enlaza con la actividad correspondiente.

GetAndSet

La clase GetAndSet implementa los métodos Get y Set de las “Flags” que se utilizan a lo largo de toda la aplicación, así como permite que los datos de temperatura y marca de tiempo sean accesibles desde cualquier parte de la aplicación.

Carpeta Libs

Incluye código JAR con librerías que se usaran en el proyecto. En la siguiente figura se muestran las librerías utilizadas en la aplicación.

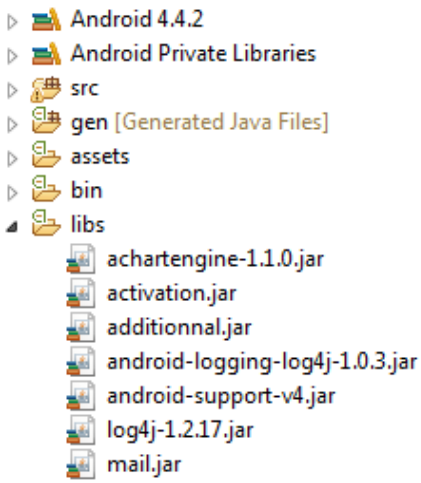


Figura 74. Vista del contenido de la carpeta Libs de la aplicación.

Librerías utilizadas en la aplicación:

Achartengine [57]: Librería necesaria para realizar el dibujado de la gráfica utilizada para la pestaña “Gráfica”.

Javamail-Android [58]: Librería utilizada para enviar correos electrónicos de forma automática sin interacción del usuario, se compone de:

-activation.jar

-additionnal.jar

-mail.jar

Log4j y android-logging-log4j-1.0.3 [59] [60]: La librería Log4j de java y la librería que permite su adaptación a Android, se utilizará para guardar en un archivo txt almacenado en la memoria del dispositivo un registro con las temperaturas recibidas y su marca de tiempo asociada

Android-support-v4: Librería incluida por defecto en todas las aplicaciones Android.

Android Manifest.xml

En este fichero se indicaran las actividades, intenciones, servicios y proveedores de contenido de la aplicación. También se declaran los permisos que requerirá la aplicación. Se indica la versión mínima de Android para poder ejecutarla, el paquete Java, la versión de la aplicación, etc.

Características más relevantes del AndroidManifest.xml de la aplicación

- Se declara la versión de API máxima y mínima soportada por la APP. En este caso la versión de API mínima será la 18, correspondiente a la versión 4.3 de Android, y la máxima la 19, que corresponde a la versión 4.4.
- Se declaran los permisos que necesita la aplicación:
    - Permiso para escribir en un archivo almacenado en la memoria del dispositivo (necesario para crear el archivo Log.txt y escribir en él desde la aplicación).
    - Permisos Bluetooth.

  - Permiso para acceder a Internet (necesario para notificaciones mediante correo electrónico).
  - Permisos Wi-fi (ver estado y cambiar el estado).
  - Permiso para enviar SMS.

- Se obliga a la aplicación a que la imagen mostrada durante la SplashScreenActivity no se muestre en la orientación horizontal (denominada *Landscape*), ya que al ser una imagen no se adapta bien y se deforma.
- Se indica que existe un servicio en la aplicación implementado en la clase ThermometerManager. En Android los servicios son componentes de la aplicación que representan operaciones que no requieren interacción por parte del usuario y que se extienden durante toda la vida de la aplicación (recibir temperatura en este caso).

4.6.6 La aplicación paso a paso

Una vez explicado que procedimientos se han llevado a cabo en el interior de la aplicación, con el fin de mostrar de manera más clara y visual el funcionamiento de la misma, se mostrará el aspecto exterior final de la aplicación en un dispositivo móvil.

Elección de la aplicación

Mediante un icono personalizado, permite al usuario diferenciar rápidamente la aplicación. Abre la aplicación.

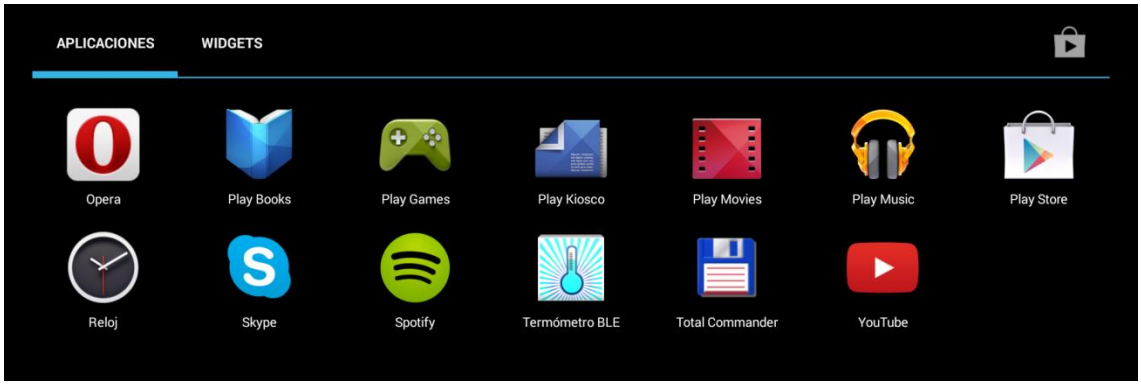


Figura 75. Elección de la aplicación.

Pantalla de inicio (SplashScreen)

Figura 76. Pantalla de inicio.

Aparece nada más entrar en la aplicación, se muestra durante 2 segundos y da paso a la vista principal.

Activación del Bluetooth (1)

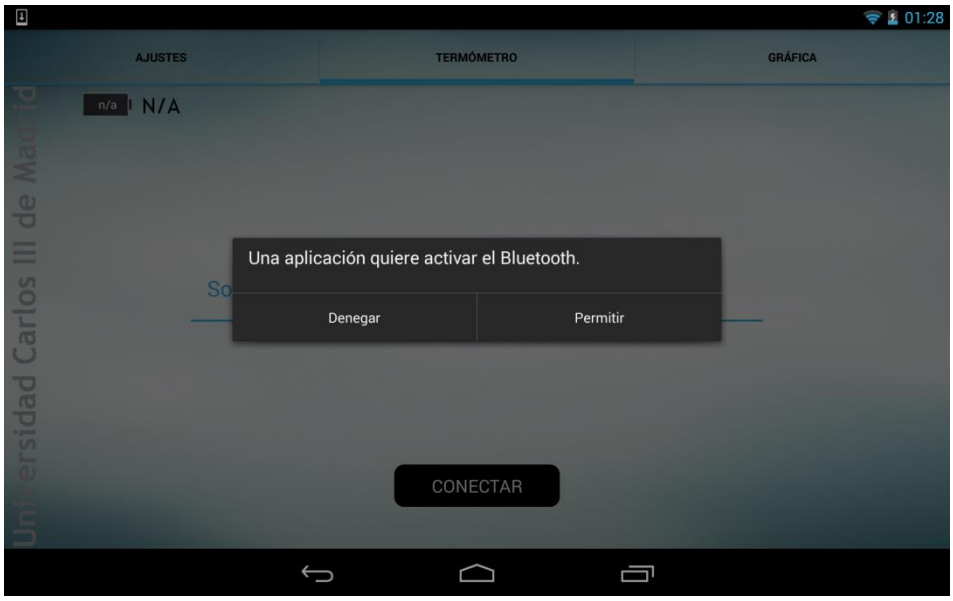


Figura 77. Activación del Bluetooth (1).

En el caso de detectar que el Bluetooth (BLE) se encuentra desactivado, muestra una ventana para activarlo. En el caso de que se denegase la activación del Bluetooth, la ventana reaparecería cada vez que el usuario quisiese conectar con el termómetro BLE.

Activación del Bluetooth (2)

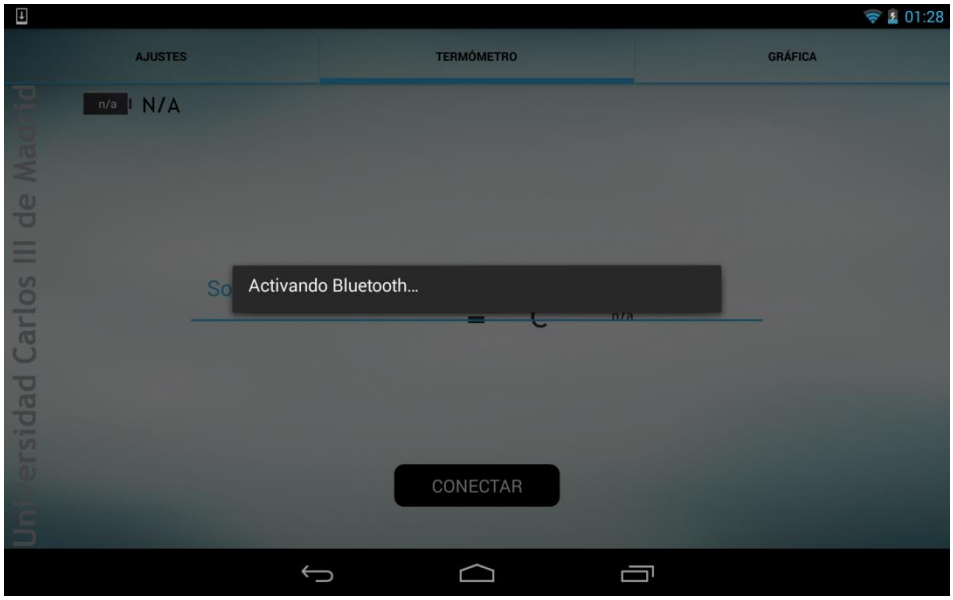


Figura 78. Activación del Bluetooth (2).

Selección de dispositivo (1)

Figura 79. Ventana de selección de dispositivo (estado inicial).

En el caso de encontrar dispositivo, aparecerá el nombre configurado, la potencia de la señal recibida (RSSI) y la dirección MAC.

Figura 80. Ventana de selección de dispositivo (dispositivo detectado).

Mostrando datos en la pestaña Termómetro (1)

Muestra la temperatura y cuando fue recibida (marca de tiempo). La marca de tiempo es mostrada en formato Año – día – mes / hora:minutos:segundos / zona horaria.

Figura 81.Vista de la pestaña Termómetro recibiendo y mostrando datos.

Mostrando datos en la pestaña Termómetro (2)

Si la temperatura recibida excede los límites del sensor TC74 (-40°C, +125°C) será considerada como no válida y la aplicación mostrara un mensaje alertando al usuario de que algo está fallando por parte del hardware del sistema (figura 82). Se ha elegido mantener los valores límite del sensor ya que así se cubre también el supuesto de una posible desconexión del sensor de temperatura, bajo la cual el sistema envía un valor de temperatura excesivamente grande (mayor en varios ordenes de magnitud al límite superior del sensor).

Figura 82. Error en los valores recibidos de temperatura.

Pestaña Ajustes

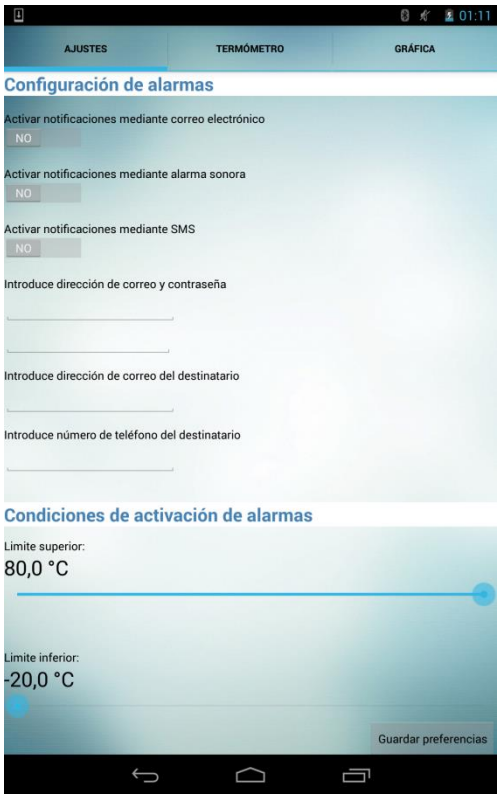


Figura 83.Vista de la pestaña Ajustes.

En esta pestaña será donde el usuario podrá configurar todas las notificaciones y las condiciones de activación de las alarmas.

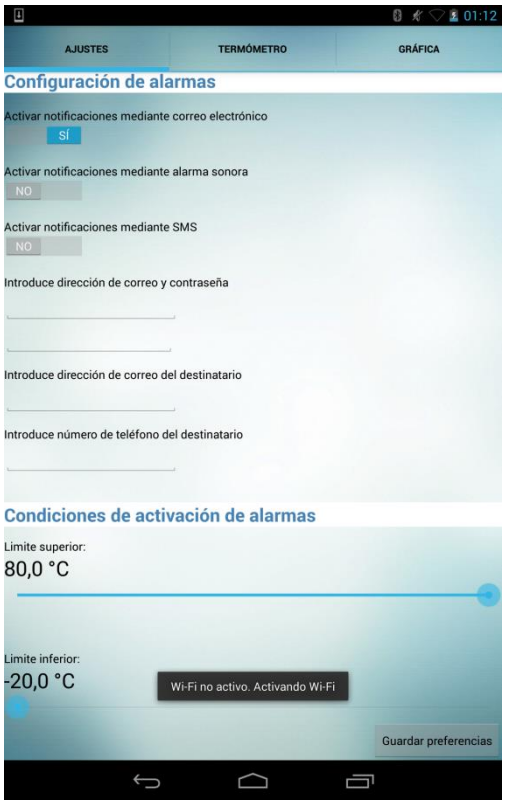


Figura 84. Activando las notificaciones mediante correo electrónico.

Antes de activar las notificaciones mediante correo electrónico la aplicación siempre comprueba que esté disponible una conexión a Internet, activando el Wi-Fi en caso contrario.

Así mismo, la aplicación comprueba en un hilo ejecutado en segundo plano si el valor de temperatura recibido excede alguno de los límites establecidos por el usuario. En caso de que así sea, se notifica a los usuarios a través de las vías configuradas por el usuario de la aplicación.

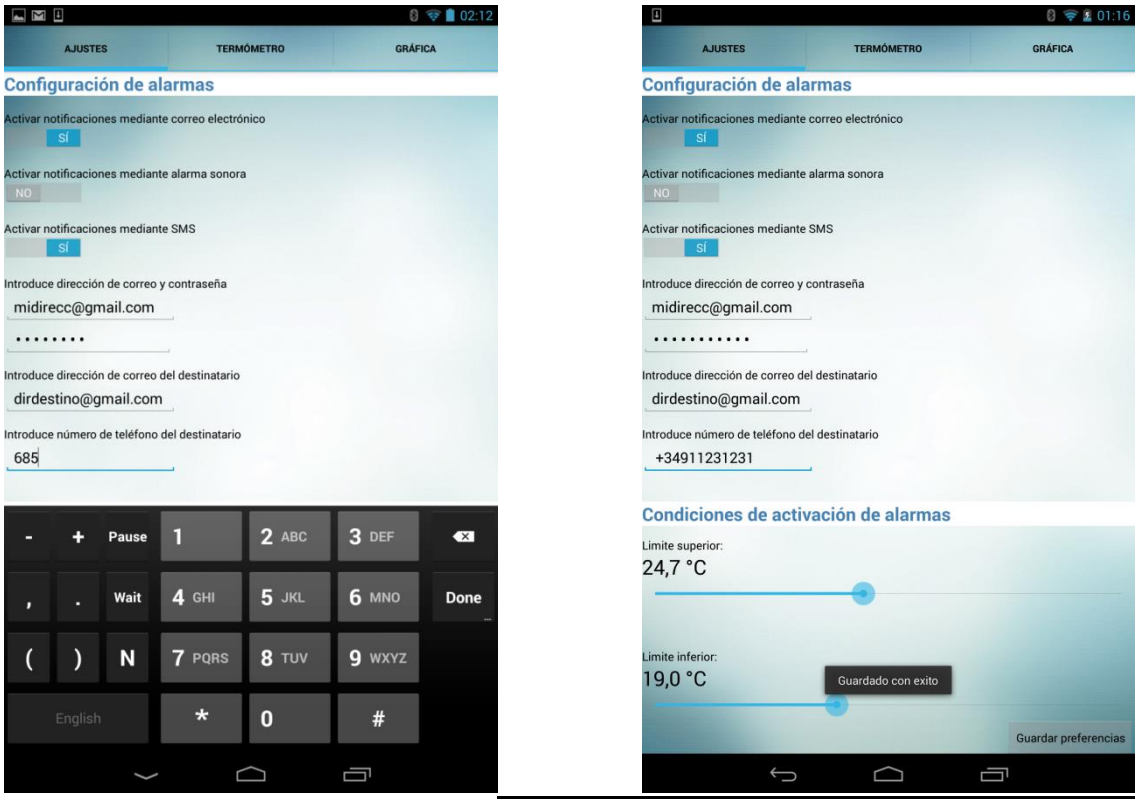


Figura 85 (izquierda) Introduciendo el número de teléfono al que notificar vía SMS en caso de alarma. Figura 86 (derecha) Guardado de preferencias.

En la figura 85 se puede apreciar el menú al que se puede acceder para introducir el número de teléfono al cual se enviara la notificación vía SMS. Una vez finalizado el proceso de configuración es posible guardar los cambios realizados (figura 86), para que estén disponibles cada vez que el usuario utilice la aplicación.

Notificación mediante correo electrónico

El usuario indicado en el campo “Introduce dirección de correo del destinatario” recibirá en su buzón de entrada un mensaje cuando la temperatura recibida exceda alguno de los límites establecidos.

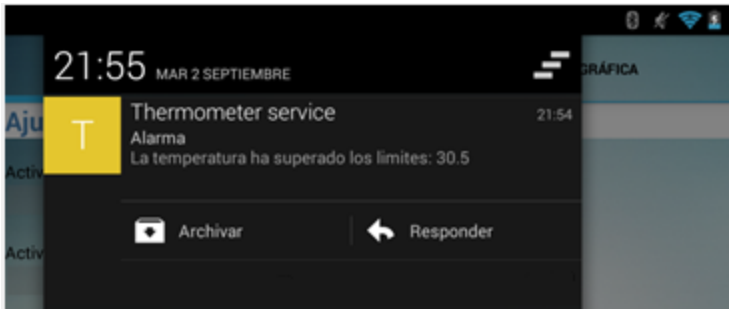


Figura 87. Recepción de notificación vía e-mail.

Notificación mediante SMS

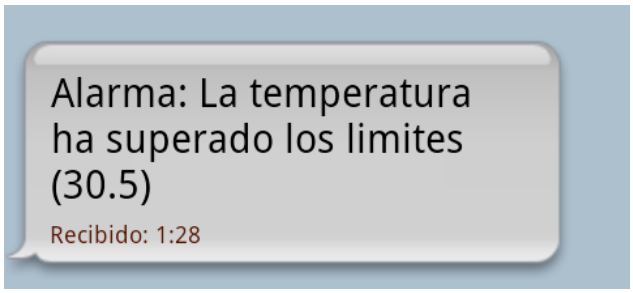


Figura 88. Recepción de notificación vía SMS.

El número indicado en el campo “Introduce número de teléfono del destinatario” recibirá un SMS cuando la temperatura recibida exceda alguno de los límites establecidos.

Pestaña Gráfica

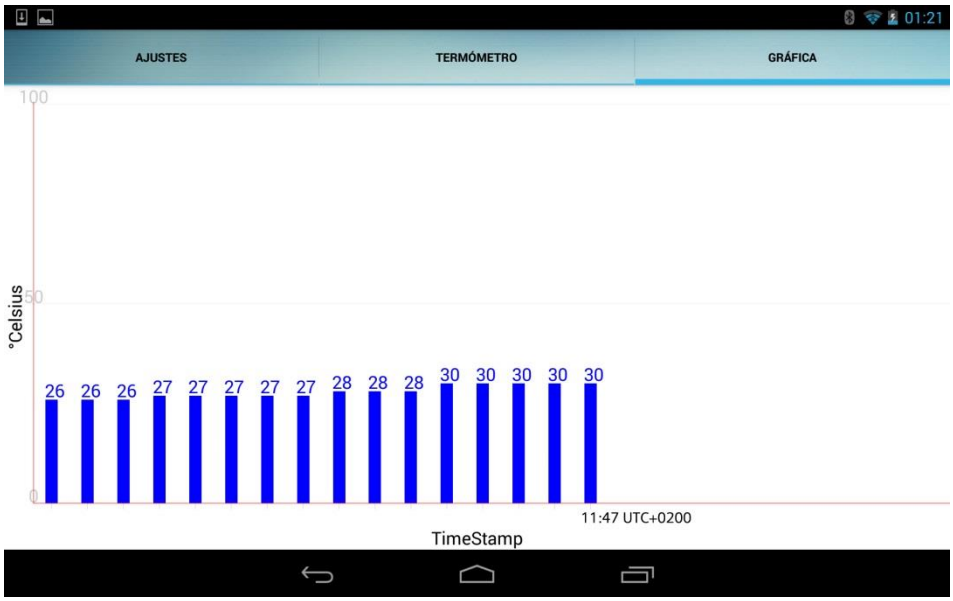


Figura 89. Vista de la pestaña Gráfica.

La aplicación actualiza mediante un hilo ejecutado en segundo plano la información mostrada en el gráfico. Ambos hilos ejecutados en segundo plano (el relativo a la pestaña “Ajustes” y el de la pestaña “Gráfica”) son ejecutados de forma paralela y de manera independiente a la pestaña por la que se encuentre navegando el usuario.

Log de la aplicación

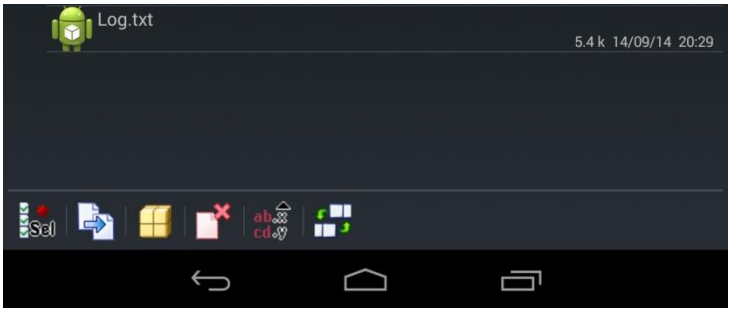


Figura 90. Archivo creado por la aplicación en la memoria del dispositivo.

El archivo “Log.txt” creado por la aplicación almacena cada medición de temperatura y su marca de tiempo asociada en orden cronológico. A continuación se muestra un ejemplo de una entrada del archivo:

2014-09-14 20:27:01,996 - [INFO::android.util.Log::bletd.Thermometer.ThermometerActivity$1] – Temperatura: 26.0 | Timestamp: 2014-14-09 20:27:01 UTC+0200 –

Donde Temperatura muestra el valor recibido en grados Celsius y Timestamp muestra la marca de tiempo asociada en formato Año-Dia-Mes / Hora:Minutos:Segundos / Zona Horaria. La zona horaria indicada en este caso es UTC (del término inglés Coordinated Universal Time) + 2, lo que equivale al Horario Europeo Central de Verano.

5 Evaluación y validación del sistema

A la hora de realizar una evaluación del sistema se llevaron a cabo una serie de pruebas para comprobar el correcto funcionamiento del mismo bajo diferentes condiciones de entorno. Todas las pruebas se han realizado utilizando el sistema diseñado y un dispositivo remoto en el cual se ha instalado la aplicación Android para monitorizar el sistema (Tableta Google Nexus 7, segunda generación). A continuación se presenta una muestra de los resultados obtenidos en algunas de las pruebas realizadas.

5.1 Pruebas de rendimiento

A continuación se realiza una descripción de las pruebas de rendimiento que fueron realizadas sobre el sistema, así como el resultado obtenido.

Prueba de rendimiento 1: Entorno a baja temperatura.

Descripción de la prueba: Se ubica el sistema de medición en un entorno cerrado (nevera) midiendo el tiempo que tarda en conseguir una medición estable de temperatura así como analizando gráficamente el comportamiento del sistema bajo estas condiciones.

Figura 91. Izquierda temperatura congelador (-20°C), derecha temperatura nevera (4°C).

Condiciones de la prueba: El sensor pasa de estar a una temperatura ambiente de 25°C a estar en un entorno cerrado a 4°C.

Figura 92. Dispositivo en entorno cerrado a 4°C.

Resultados:

Una vez ubicado el sistema en el entorno a 4°C el tiempo que tarda el sistema en conseguir una medición estable de temperatura es de aproximadamente 1500 segundos. A partir de la estabilización de la temperatura esta se mantiene constante durante el resto de la prueba.

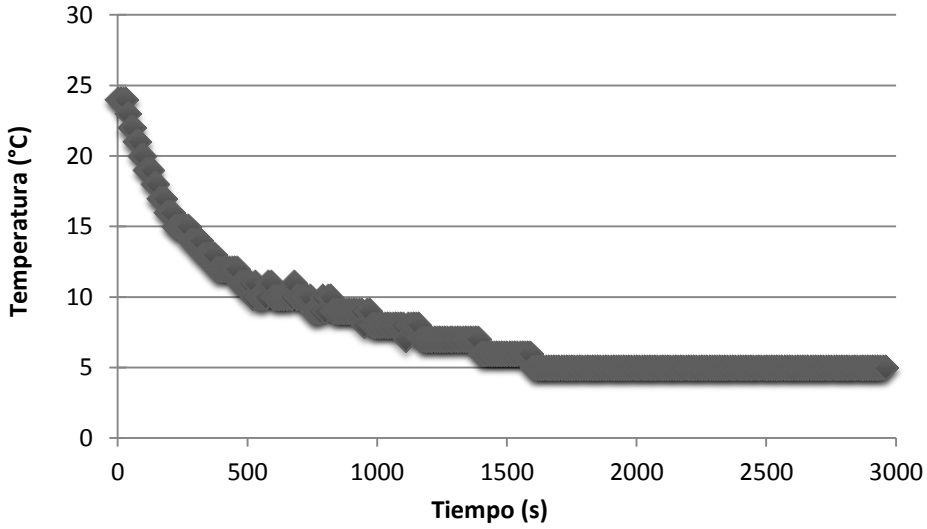


Figura 93. Evolución de la temperatura medida por el sensor a lo largo del tiempo a baja temperatura.

Prueba de rendimiento 2: Entorno a temperatura media.

Descripción de la prueba: Se ubica el sistema de medición en un entorno abierto a 25 °C (interior vivienda) midiendo el tiempo que tarda en conseguir una medición estable de temperatura así como analizando el comportamiento del sistema bajo estas condiciones.

Figura 94. Dispositivo en entorno abierto a 25°C.

Condiciones de la prueba: El sensor pasa de estar en un entorno a una temperatura ambiente de 31°C a un entorno a 25°C.

Resultados:

Una vez ubicado el sistema en el entorno a 25°C el tiempo que tarda el sistema en conseguir una medición estable de temperatura es de aproximadamente 600 segundos. A partir de la estabilización de la temperatura esta se mantiene constante durante el resto de la prueba.

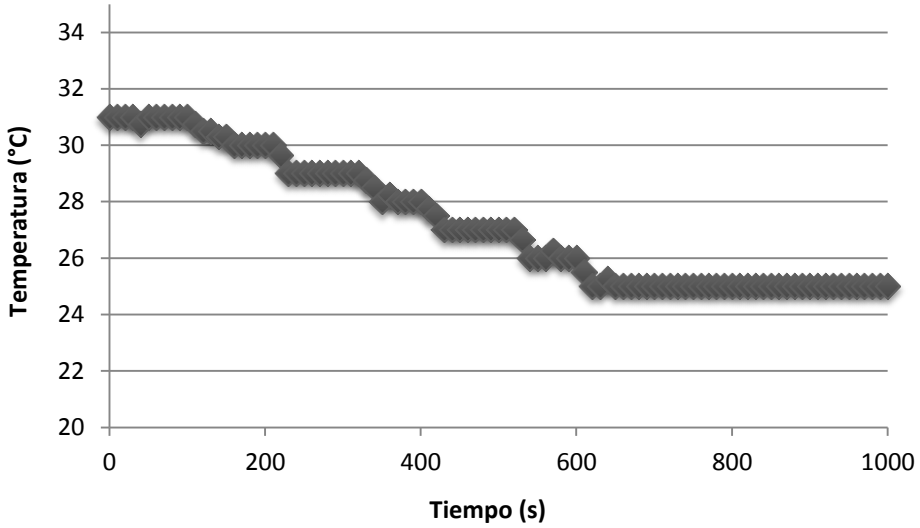


Figura 95. Evolución de la temperatura medida por el sensor a lo largo del tiempo a temperatura media.

Prueba de rendimiento 3: Entorno a alta temperatura.

Descripción de la prueba: Se simula un entorno cerrado a 50 °C utilizando un soldador eléctrico aplicado directamente sobre el sensor de temperatura, midiendo el tiempo que tarda en conseguir una medición estable de temperatura así como analizando el comportamiento del sistema bajo estas condiciones de entorno.

Condiciones de la prueba: El sensor pasa de estar a una temperatura ambiente de 26°C a unas condiciones equivalentes a un entorno cerrado a 50°C.

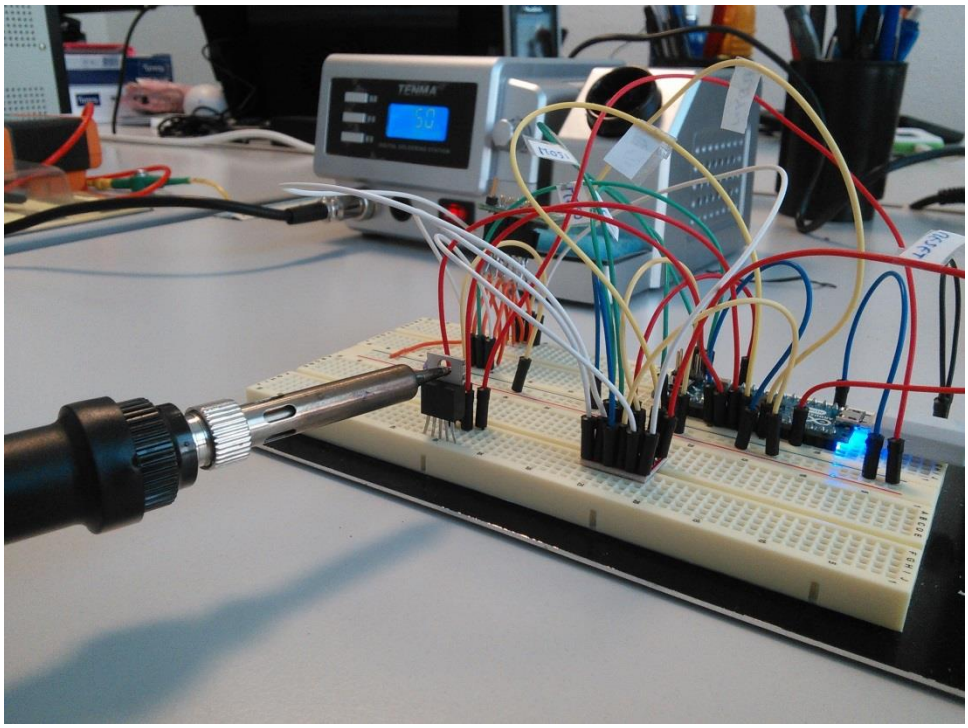


Figura 96. Soldador a 50°C aplicado sobre el sensor de temperatura.

Resultados:

Una vez aplicado el soldador a 50°C sobre el sensor de temperatura el tiempo que tarda el sistema en conseguir una medición estable de temperatura es de aproximadamente 400 segundos. A partir de la estabilización de la temperatura esta se mantiene constante durante el resto de la prueba.

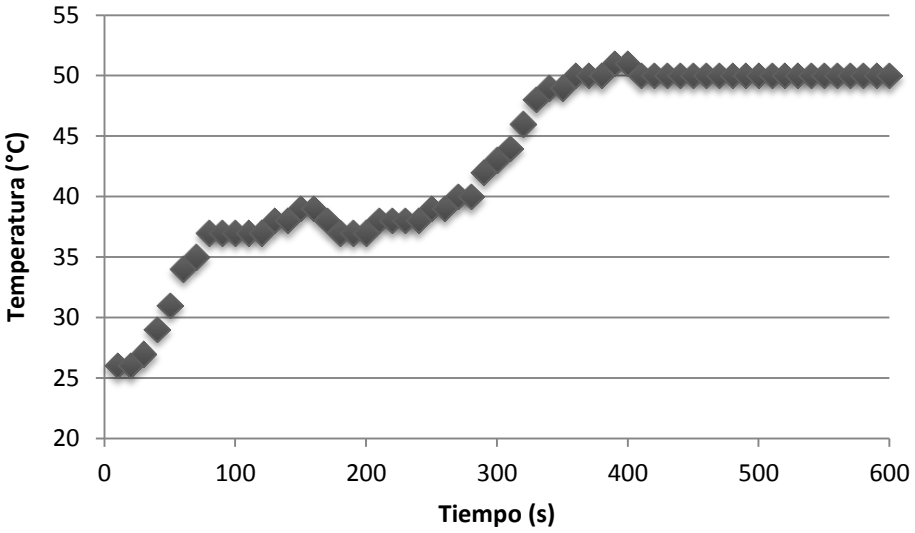


Figura 97. Evolución de la temperatura medida por el sensor a lo largo del tiempo a alta temperatura.

5.2 Pruebas de batería y consumo

El sistema consume corriente de la fuente de alimentación externa en función del estado en el que se encuentre: Realizando *advertising*, conectado con un dispositivo remoto o en modo *Sleep.* Para realizar los cálculos relativos a cada prueba es necesario conocer el consumo de corriente en cada estado. En primer lugar se mide la corriente consumida por el sistema cuando el chip nRF8001 se encuentra realizando *advertising.*

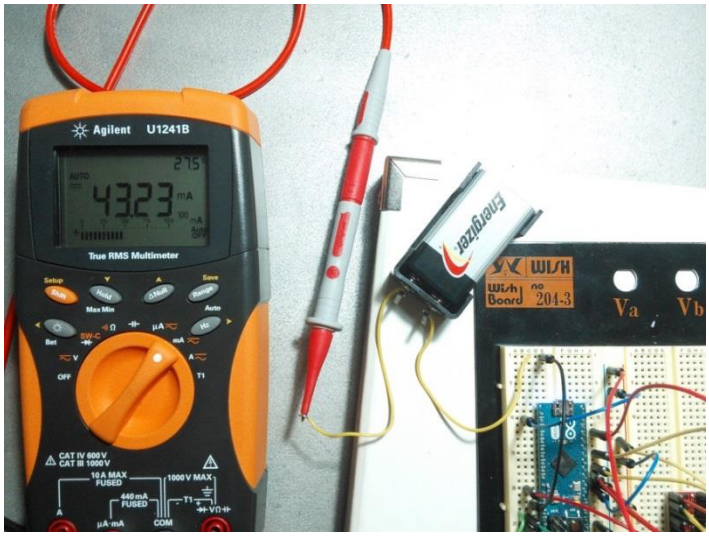


Figura 98. Consumo de corriente del sistema realizando *advertising*.

En segundo lugar se mide la corriente consumida por el sistema cuando existe una conexión activa con un dispositivo remoto.

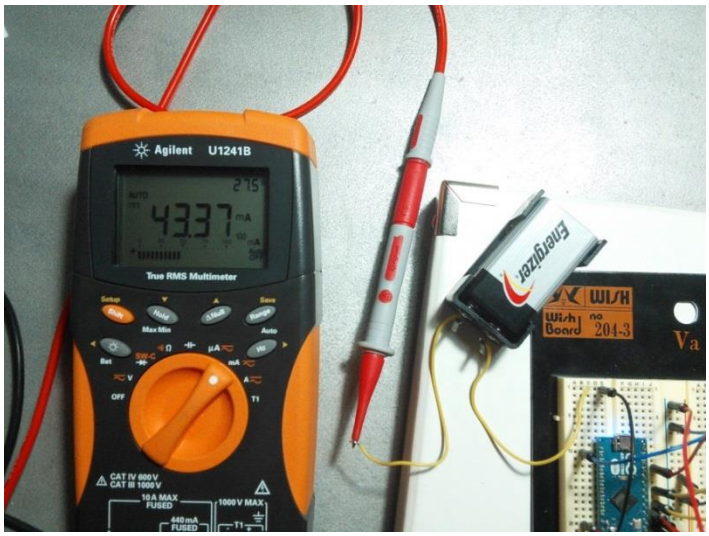


Figura 99. Consumo de corriente del sistema cuando se encuentra conectado a un dispositivo remoto.

Como se puede apreciar la diferencia es inferior a 1 mA llegando en ocasiones a igualarse, esto se debe a que el consumo de corriente del chip nRF8001 realizando *advertising* y el consumo cuando existe una conexión activa con un dispositivo remoto dependen directamente del número de transmisiones y recepciones realizadas por el chip [37], el cual varía en cada comunicación.

Por último se mide la corriente consumida por el sistema cuando este se encuentra en modo *Sleep*.

Figura 100. Consumo de corriente del sistema cuando se encuentra en modo *Sleep*.

Prueba de batería y consumo 1: Uso intensivo.

Descripción de la prueba: Se analiza el consumo de batería del sistema de medición con unos parámetros de uso definidos:

- Tiempo que el sistema se encuentra realizando *advertising*: 2,5% del total.

- Tiempo que el sistema se encuentra en estado *Connected*: 92,5% del total.

- Tiempo que el sistema se encuentra en modo *Sleep*: 5% del total.

Condiciones de la prueba: El sensor se ubica en un entorno a una temperatura ambiente de 26°C.

Consumo de corriente del sistema:

$$(43{,}23mA*1{,}5min+43{,}37mA*55{,}5min+14{,}2mA*3min)/60min = 41{,}9\ mA$$

Utilizando una batería de 9v de capacidad 595mAh como fuente de alimentación externa, la duración será de:

$$595mAh/41{,}9mA \approx 14\ horas.$$

Prueba de batería y consumo 2: Uso moderado.

Descripción de la prueba: Se analiza el consumo de batería del sistema de medición con unos parámetros de uso definidos:

- Tiempo que el sistema se encuentra realizando *advertising*: 20% del total.
- Tiempo que el sistema se encuentra en estado Connected: 40% del total.
- Tiempo que el sistema se encuentra en modo Sleep: 40% del total.

Condiciones de la prueba: El sensor se ubica en un entorno a una temperatura ambiente de 26°C.

Consumo de corriente del sistema:

$$(43{,}23mA*12min+43{,}37mA*24min+14{,}2mA*24min)/60min = 31{,}67\ mA$$

Utilizando una batería de 9v de capacidad 595mAh como fuente de alimentación externa, la duración será de:

$$595mAh/31{,}67mA \approx 19\ horas.$$

Prueba de batería y consumo 3: Uso reducido.

Descripción de la prueba: Se analiza el consumo de batería del sistema de medición con unos parámetros de uso definidos:

- Tiempo que el sistema se encuentra realizando *advertising*: 31,66% del total.

- Tiempo que el sistema se encuentra en estado Connected: 5% del total.

- Tiempo que el sistema se encuentra en modo Sleep: 63,33% del total.

Condiciones de la prueba: El sensor se ubica en un entorno a una temperatura ambiente de 26°C.

Consumo de corriente del sistema:

$$(43{,}23mA*19min+43{,}37mA*3min+14{,}2mA*38min)/60min = 24{,}85\ mA$$

Utilizando una batería de 9v de capacidad 595mAh como fuente de alimentación externa, la duración será de:

$$595mAh/24{,}85\ mA \approx 24\ horas.$$

5.3 Pruebas de precisión

Las pruebas de precisión realizadas se basarán en analizar la precisión de las medidas de temperatura obtenidas por el sistema y su posterior comparación con la medida obtenida por otra fuente fiable.

Prueba de precisión 1: Temperaturas en el rango de +25°C a +85°C.

Descripción de la prueba: Analizando la hoja de características del sensor de temperatura utilizado [31], se comprueba que este presenta una precisión en las medidas variable en función del rango de temperaturas en que se encuentre. En esta prueba se analizará la precisión de las mediciones obtenidas en el rango de +25°C a +85°C, comparando el valor medido con el de varias fuentes durante 1 hora.

Temperatura medida por otras fuentes (valor de referencia): 31°C.

Resultados:

De las 353 mediciones realizadas durante la prueba, se obtuvieron 4 medidas de 30,75°C, 7 medidas de 31,25°C y 342 de 31°C. Se estabiliza la medición a 31°C.

El TC74 presenta una precisión de ±2°C en el rango de +25°C a +85°C, luego los resultados obtenidos se ajustan a lo esperado (figura 101).

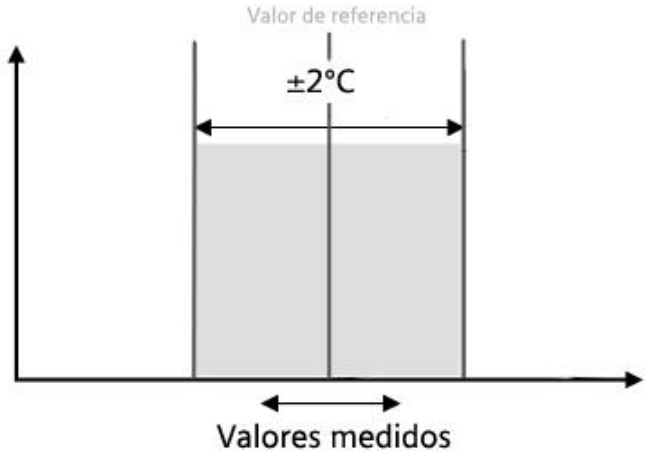


Figura 101. Precisión del sistema en el rango de +25°C a +85°C.

Prueba de precisión 2: Precisión en el rango de 0°C a +25°C.

Descripción de la prueba: Analizando la hoja de características del sensor de temperatura utilizado [31], se comprueba que este presenta una precisión en las medidas variable en función del rango de temperaturas en que se encuentre. En esta prueba se analizará la precisión de las mediciones obtenidas en el rango de 0°C a +25°C, comparando el valor medido con el de una fuente fiable durante 1 hora.

Temperatura medida por otras fuentes (valor de referencia): 4°C.

Resultados:

De las 350 mediciones realizadas durante la prueba, se obtuvieron 9 medidas de 5,25°C y 341 de 5°C. Se estabiliza la medición a 5°C.

El TC74 presenta una precisión de ±3°C en el rango de 0°C a +25°C, luego los resultados se ajustan a lo esperado (figura 102).

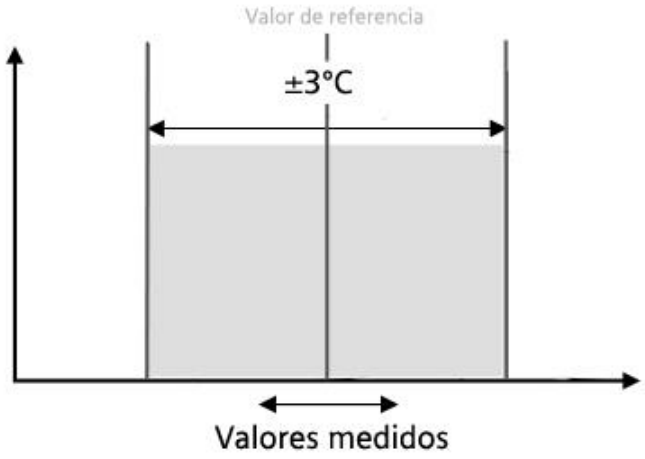


Figura 102. Precisión del sistema en el rango de +0°C a +25°C.

5.4 Pruebas de alcance

El alcance de las comunicaciones mediante Bluetooth low energy del sistema viene limitado por la antena PCB del módulo BLE nRF2740. Según las pruebas realizadas tanto en campo abierto (sin obstáculos entre el sistema y el dispositivo remoto), como en interiores (con obstáculos entre el sistema y el dispositivo remoto), este se encuentre en un rango estable de 10 metros, pudiendo ser superior bajo determinadas condiciones.

En la figura 103 se presenta un gráfico que muestra el valor de la potencia recibida por la radio del dispositivo remoto en dBm. Los valores recibidos varían desde los 0 dBm (1000 µW) hasta valores inferiores a los -90 dBm (1 pW).

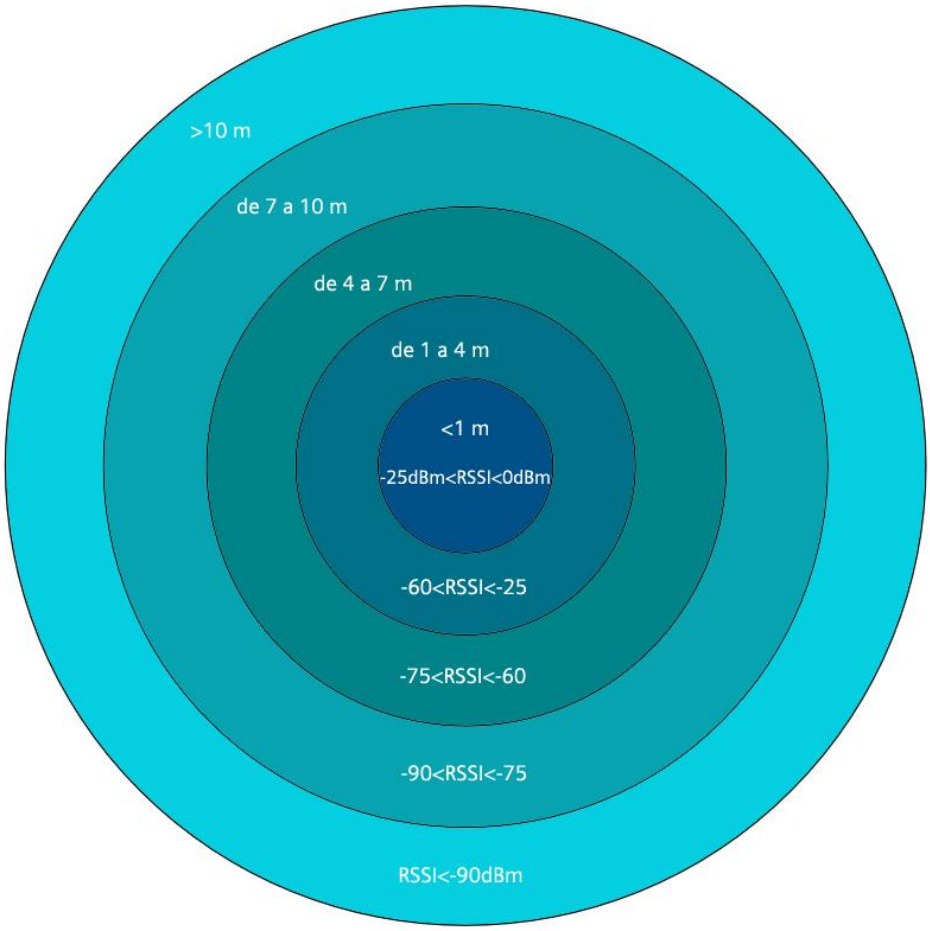


Figura 103. Potencia recibida por la radio del dispositivo remoto en dBm en función de la distancia entre el sistema y el dispositivo.

6 Planificación y fases de desarrollo

A continuación se describen cada una de las fases generales presentes en la elaboración del trabajo, así como un diagrama Gantt en el que se representan gráficamente cada una de ellas junto a las tareas que las conforman.

6.1 Fases del desarrollo

- Análisis de la tecnología. Se analizan las posibilidades de la tecnología disponible, analizando de manera individual cada componente del sistema, el software a utilizar y las posibilidades de los diferentes sistemas operativos a la hora de programar la aplicación móvil.

- Definición del sistema: Se definen las funcionalidades del sistema así como cuáles son las restricciones sobre las que se va a trabajar.

- Aprendizaje: Se adquieren los conocimientos extra necesarios para desarrollar el sistema y la aplicación móvil.

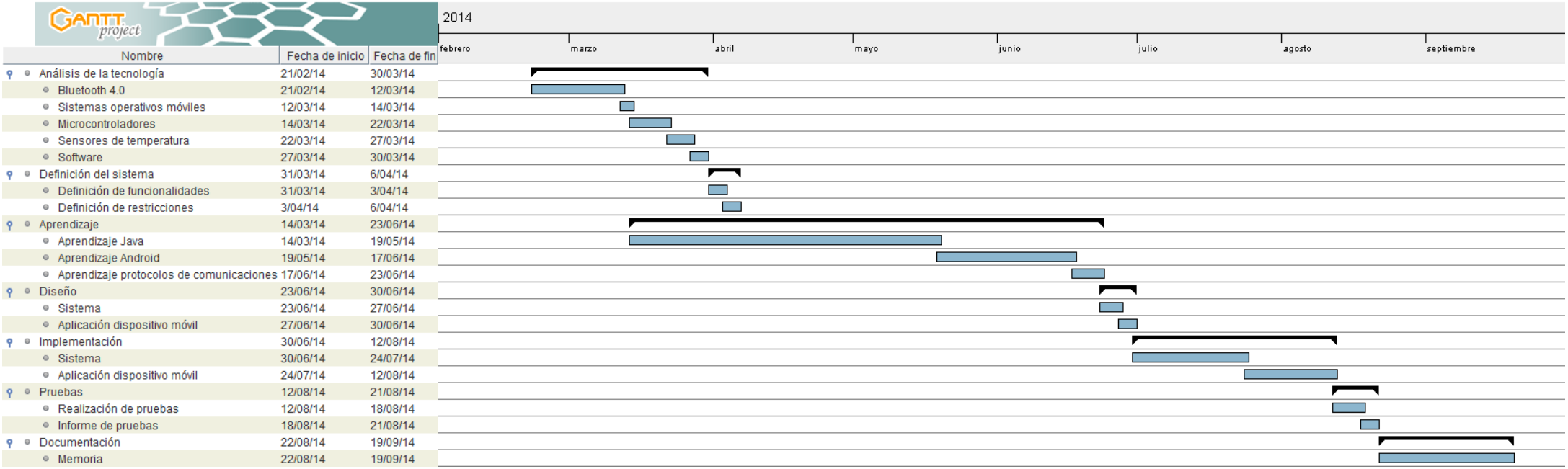
- Diseño: Se diseña el sistema y la aplicación móvil para su posterior implementación.

- Implementación: Fase de ejecución en la que se pasa del diseño a la implementación del sistema desarrollando las soluciones software y hardware necesarias.

- Pruebas: Se evalúa el comportamiento del sistema bajo diferentes condiciones de entorno, adquiriendo y posteriormente analizando información sobre el comportamiento real del mismo.

- Documentación: Se redacta la memoria recopilando detalles sobre todo el trabajo realizado.

6.2 Diagrama de Gantt



Nombre	Fecha de inicio	Fecha de fin
Análisis de la tecnología	21/02/14	30/03/14
Bluetooth 4.0	21/02/14	12/03/14
Sistemas operativos móviles	12/03/14	14/03/14
Microcontroladores	14/03/14	22/03/14
Sensores de temperatura	22/03/14	27/03/14
Software	27/03/14	30/03/14
Definición del sistema	31/03/14	6/04/14
Definición de funcionalidades	31/03/14	3/04/14
Definición de restricciones	3/04/14	6/04/14
Aprendizaje	14/03/14	23/06/14
Aprendizaje Java	14/03/14	19/05/14
Aprendizaje Android	19/05/14	17/06/14
Aprendizaje protocolos de comunicaciones	17/06/14	23/06/14
Diseño	23/06/14	30/06/14
Sistema	23/06/14	27/06/14
Aplicación dispositivo móvil	27/06/14	30/06/14
Implementación	30/06/14	12/08/14
Sistema	30/06/14	24/07/14
Aplicación dispositivo móvil	24/07/14	12/08/14
Pruebas	12/08/14	21/08/14
Realización de pruebas	12/08/14	18/08/14
Informe de pruebas	18/08/14	21/08/14
Documentación	22/08/14	19/09/14
Memoria	22/08/14	19/09/14

Figura 104. Diagrama de Gantt.

7 Conclusiones y líneas futuras

7.1. Conclusiones

En este trabajo, una vez efectuado el diseño y posterior implementación del sistema, analizando las consideraciones experimentales y operativas de cada parte y proceso desde el conocimiento del comportamiento real de la aplicación, es hora de presentar las conclusiones obtenidas.

El enfoque planteado para la presentación de estas conclusiones es el de relacionarlas con los objetivos iniciales del trabajo, y con las líneas futuras a seguir con el fin de perfeccionar el sistema diseñado.

Se ha logrado desarrollar un sistema de monitorización de temperatura capaz de comunicar de forma inalámbrica mediante Bluetooth low energy con cualquier dispositivo Android que disponga de un nivel API igual o superior a 18. Se ha diseñado y desarrollado la interfaz hardware para realizar las comunicaciones dentro del sistema, así como una interfaz software para controlarlas.

El sistema desarrollado abarca desde la utilización de protocolos de comunicación en serie, SPI e I^2C, protocolos de comunicaciones inalámbricas, Bluetooth low energy, hasta aplicaciones desarrolladas para sistemas operativos móviles, en este caso Android. Dado el carácter multidisciplinar del sistema implementado, su desarrollo ha

requerido un proceso de aprendizaje y mejora constante llevado a cabo de forma paralela a la realización del presente trabajo.

Las pruebas realizadas han proporcionado datos reales que contribuyen a la validación del sistema completo así como se ha llevado a cabo un estudio del alcance de las comunicaciones inalámbricas para conocer las posibilidades de esta arquitectura.

En conjunto, el sistema desarrollado se define como un sistema abierto a modificaciones, mostrando una *escalabilidad* inherente a su diseño modular que impone límites reducidos a posibles ampliaciones.

7.2 Líneas futuras

A lo largo de la realización de este trabajo han surgido múltiples opciones a la hora de diseñar e implementar el sistema realizado. La solución final planteada es el resultado de todas las decisiones individuales tomadas a lo largo de este proceso. Sin embargo, en el camino han quedado algunas alternativas que presentan características muy interesantes que no han podido ser explotadas en su totalidad. A continuación se muestran algunas de ellas:

• Incremento del número de sensores utilizados. Implementar un sistema con múltiples sensores de temperatura que, aprovechando las funcionalidades del bus I^2C, permita al sistema desde realizar múltiples medidas en diferentes zonas de manera simultánea a contrastar repetidas veces la temperatura medida en un mismo escenario con diferentes sensores.

• Mejora de la carcasa. En este campo son muchas las posibles mejoras a realizar, estando en última instancia únicamente condicionadas al presupuesto disponible. Proteger al sistema de la condensación, permitiendo su uso en ambientes húmedos o a bajas temperaturas, u optimizar la carcasa reduciendo su tamaño y reubicando los componentes son solo una muestra de las posibles opciones disponibles.

• Utilización de una antena externa que mejore el alcance de las comunicaciones vía Bluetooth low energy. Si bien el alcance obtenido por el sistema entra dentro de lo

esperado según lo establecido por la especificación Bluetooth [2], un mayor alcance permitiría mayor libertad de movimiento al usuario.

• Duración de la batería: Con unos consumos de corriente del orden de los μA tanto en sensor de temperatura como en módulo Bluetooth, una batería de 595 mAh debería durar varios meses. Sin embargo, la placa de desarrollo presenta un consumo elevado en comparación con el resto de componentes, por lo que una mejora sustancial en la autonomía del sistema pasaría por eliminar la placa de desarrollo y buscar alternativas a la utilización de la misma como pieza central.

• Utilización de sensores de diferentes tipos. Esta línea de mejora requiere ajustes en diferentes partes del sistema, principalmente por el lado del chip nRF8001 y del dispositivo remoto ya que estos están preparados para enviar (recibir en el segundo caso) únicamente mediciones de temperatura. Sin embargo, abre múltiples posibilidades al sistema diseñado, aprovechando la capacidad del microcontrolador para crear un sistema más completo y que funcione en más escenarios que el diseñado.

• Uso de *fragments* a la hora de implementar la UI de la aplicación Android. Una solución más dinámica y completa que ha reemplazado al uso de herramientas "*Tab" (TabHost, TabActivity y TabWidget)* en la mayoría de interfaces graficas con múltiples paneles. Un ejemplo de su uso es la aplicación Play Store de Google, programada utilizando *fragments.*

Referencias

[1] http://en.wikipedia.org/wiki/Internet_of_Things [Online]

[2] Specification of the Bluetooth System, Covered Core Package, Version: 4.0 & 4.1. The Bluetooth Special Interest Group.

[3] http://www.nobelprize.org/nobel_prizes/physics/laureates/1956 [Online]

[4] The Most Widely Used Computer on a Chip: The TMS 1000. State of the Art: A Photographic History of the Integrated Circuit. Augarten, Stan. 1983.

[5] http://smithsonianchips.si.edu/ice/cd/MEMORY97/SEC09.PDF. 9 rom, eprom, and eeprom technology. [Online]

[6] http://www.icinsights.com/news/bulletins/MCU-Market-On-Migration- Path-To-32bit-And-ARMbased-Devices [Online]

[7] http://www.gadgetselectroniques.com/Bluetooth-Thermometer-4-Android-bbxaaaSCa.asp [Online]

[8] http://thermometer.co.uk/bluetooth-temperature-probes/1002-bluetooth-thermometer-bluetherm-duo.htm [Online]

[9] The key for Bluetooth low energy technology. West, Alex.

[10] Introducing Bluetooth Smart: Part 1: a Look at Both Classic and New Technologies. DeCuir, J. IEEE Consumer Electronics Magazine Volume: 3 Issue: 1 Pages: 12-18. Enero 2014.

[11] Overview and Evaluation of Bluetooth Low Energy: An Emerging Low-Power Wireless Technology. Carles Gomez , Joaquim Oller y Josep Paradells. Sensors 2012, 12, 11734-11753

[12] Wireless home automation networks for indoor surveillance: technologies and experiments. Milo Spadacini; Stefano Savazzi; Monica Nicoli. EURASIP Journal on Wireless Communications and Networking 2014, 2014:6 doi:10.1186/1687-1499-2014-6.

[13] Bluetooth LE Finds Its Niche. Want, R (Want, Roy); Schilit, B (Schilit, Bill); Laskowski, D (Laskowski, Dominik)**.** IEEE PERVASIVE COMPUTING Volume: 12 Issue: 4 Pages: 12-16. Octubre 2013.

[14] A Smart Bluetooth-Based Ad Hoc Management System for Appliances in Home Environments. Sendra, S.; Laborda, A.; Diaz, J.R.; Lloret, J. Ad-hoc, Mobile, and Wireless Networks. 13th International Conference, ADHOC-NOW 2014.

[15] http://blog.bluetooth.com/analysts-project-bluetooth-smart-technology-to-lead-home-automation-market [Online]

[16] Towards the era of wireless keys: How the IoT can change authentication paradigm. Petrov, V.; Edelev, S.; Komar, M.; Koucheryavy, Y. 2014 IEEE World Forum on Internet of Things (WF-IoT).

[17] Transmission of ECG data with the patch-type ECG sensor system using Bluetooth Low Energy. Young-jin Park ; Hui-sup Cho**.** ICT Convergence (ICTC), 2013 International Conference on Digital Object Identifier: 10.1109/ICTC.2013.6675359 Page(s): 289 – 294. 2013.

[18] A Wireless EMG recording system and its application to muscle fatigue detection. Chang, K.-M.; Liu, S.-H.; Wu, X.-H. Sensors 2012, 12, 489–499.

[19] Collaborative processing of wearable and ambient sensor system for blood pressure monitoring. Nakamura, M.; Nakamura, J.; Lopez, G.; Shuzo, M.; Yamada, I. Sensors 2011, 11, 6760–6770.

[20] Applications, challenges, and prospective in emerging body area networking technologies. Patel, M.; Wang, J. IEEE Commun. Mag. 2010, 17, 80–88.

[21] Continua: An interoperable personal healthcare ecosystem. Carroll, R.; Cnossen, R.; Schnell, M.; Simons, D. IEEE Pervasive Comput. 2007, 6, 90–94.

[22] Bluetooth low energy: Wireless connectivity for medical monitoring. Helge Omre, A. J. Diabetes Sci. Technol. 2010, 4, 457–463.

[23] https://www.fitbit.com [Online]

[24] http://www.kensington.com [Online]

[25] www.thetileapp.com [Online]

[26] www.sticknfind.com [Online]

[27] http://world.casio.com/news/2011/watch_prototype/ [Online]

[28] Is Near-Field Communication Close to Success? Ortiz, S. IEEE Comput. 2006.

[29] Curso Android: Introducción a la programación. UPVX

[30] http://www.i2c-bus.org [Online]

[31] Tiny Serial Digital Thermal Sensor (TC74) Datasheet. Microchip Technology Inc.

[32] http://www.nxp.com/documents/user_manual/UM10204.pdf [Online]

[33] Two Wires and 30 Years: A Tribute and Introductory Tutorial to the I2C Two-Wire Bus. Corcoran, Peter. IEEE consumer electronics magazine, 2013.

[34] http://en.wikipedia.org/wiki/Bit_banging [Online]

[35] http://arduino.cc/en/Main/arduinoBoardMicro [Online]

[36] Hoja de características ATmega32u4. Atmel.

[37] nRF8001. Single-chip Bluetooth® low energy solution. Product Specification 1.2. Nordic Semiconductor.

[38] http://maxembedded.com/2013/11/10/serial-peripheral-interface-spi-basics/ [Online]

[39] An Introduction to (IC)-C-2 and SPI Protocols. Leens, Frederic. IEEE instrumentation & measurement magazine. 2009, ISSN:1094-6969.

[40] nRF2740 - nRFgo nRF8001 SCC module w/PCB Antenna Revision 1.0. nRF8001-DK Rev1.0. nRF2740_rev1_0.SchDoc. Nordic Semiconductor**.**

[41] ***Bluetooth Core specification*** **v4.0, Vol. 3, Part C (GAP), Sect. 9.2.4, 'General Discoverable Mode'.**

[42] ***Bluetooth Core specification*** **v4.0, Vol. 3, Part C (GAP), Sect. 9.2.3, 'Limited Discoverable Mode'.**

[43] ***Bluetooth Core specification*** **v4.0, Vol. 3, Part C (GAP), Sect. 9.2.2, 'Non-Discoverable Mode'.**

[44] ***Bluetooth Core specification*** **v4.0, Vol. 3, Part G (GATT), Sect. 4.4.2, 'Discover Primary Service by Service UUID'.**

[45] ***Bluetooth Core specification*** **v4.0, Vol. 3, Part G (GATT), Sect. 4.6.1, 'Discover All Characteristics of a Service'.**

[46] ***Bluetooth Core specification*** **v4.0, Vol. 3, Part G (GATT), Sect. 4.7.1, 'Discover All Characteristics Descriptors'.**

[47] ***Bluetooth Core specification*** **v4.0, Vol. 3, Part G (GATT), Sect. 7.1, 'Service Changed'.**

[49] Bluetooth Profile Specification. ***Health Thermometer Profile*** **V10r00.** The Bluetooth Special Interest Group.

[50]https://developer.bluetooth.org/gatt/services/Pages/ServiceViewer.aspx?u=org.bluetooth.service.health_thermometer.xml [Online]

[51]https://developer.bluetooth.org/gatt/characteristics/Pages/CharacteristicViewer.aspx?u=org.bluetooth.characteristic.temperature_measurement.xml [Online]

[52] http://msdn.microsoft.com/en-us/ library/windows.devices.bluetooth.genericattributeprofile.gattcharacteristicproperties.aspx [Online]

[53] nRFgo Studio Help. Nordic Semiconductor.

[54] Personal Health Devices Transcoding_WP_V13. The Bluetooth Special Interest Group.

[55] Inside Bluetooth Low Energy . Naresh Gupta. Artech House Remote Sensing Library. 2013.

[56] https://github.com/NordicSemiconductor/ble-sdk-arduino [Online]

[57] http://www.achartengine.org/ [Online]

[58] https://code.google.com/p/javamail-android/ [Online]

[59] http://logging.apache.org/log4j/1.2/ [Online]

[60] https://code.google.com/p/android-logging-log4j/ [Online]